# File System Optimization Using Block Reorganization Techniques

by

**Sumit Narayan**

B.E., University of Madras, 2002

A Thesis

Submitted in Partial Fulfillment of the

Requirements for the Degree of

Master of Science

at the

University of Connecticut

2004

APPROVAL PAGE

Master of Science Thesis

File System Optimization Using Block Reorganization Techniques

Presented by
Sumit Narayan, B.E.

Major Advisor _____
John A. Chandy

Associate Advisor _____
Krishna Pattipati

Associate Advisor _____
Dong-Guk Shin

University of Connecticut
2004

# Acknowledgements

I would like to thank and acknowledge my advisor, *Dr. John A. Chandy* for his constant faith in me throughout the long, challenging, yet exciting work on this thesis. I am grateful to his openness to ideas, discussions and easiness to work with. I definitely couldn't have done this without his help. I am thankful to *Janardhan* and *Mike*, my co-workers at SNSL, for their cooperation and also for sharing their ideas with me.

I would also like to acknowledge

*Windsor Hsu*, for clarifying my doubts through emails.

*Daniel Ellard*, for providing me with traces in our Harvard Study.

*Raajaa Vishnu*, my dearest friend, for his valuable insights and comments on my work.

Finally, last, but most importantly, I would like to thank my parents, and my brother *Vineet* for their constant support and encouragement in all my endeavors. I am grateful to their love and understanding that has made this thesis happen.

# Contents

# List of Tables

**Table**

# List of Figures

**Figure**

# Abstract

*The slow mechanical nature of many storage devices is a major cause of concern for all. Processor speeds are increasing at the rate of 50% a year; while disk access times are able to gain only 10%. Rapid increase in the speed of microprocessors and the relative slowness of disk access times have caused file servers to fail to deliver the speed they are designed for. To attain good disk performance on servers, the I/O traffic patterns must be known, and the disk must be optimized for such patterns. A slew of optimization techniques are available, including using larger caches, better file system utilization, and various disk organization techniques. However, without an understanding of the actual disk access usage patterns, it is somewhat difficult to make decisions on which of these to implement.*

*Our thesis contributes to this area as summarized below. First, we introduce a disk tracing technique in the Linux kernel, using which we obtain disk access patterns under three different workloads. Next, using this knowledge, we analyze the implication of the workloads on disk I/O. Based on the analysis, we evoke an optimization technique that automatically replicates and reorganizes the selected "hot" disk blocks depending on their usage patterns. In our thesis, we suggest usage of multiple zones over the disk, instead of conventional single "hot" zone idea. Using multiple zones and relocating high-heat blocks into their respective zones, we were successful in establishing up to 10–20% additional improvement in disk performance over the already existing disk reorganization techniques, and up to 50% improvement against the unaltered disk layout. Further simulations demonstrate similar performance improvement under RAID systems.*

*To my family...*

# Chapter 1

# Introduction

## 1.1 Motivation

The slow mechanical nature of many storage devices is a major cause of concern for all. With the processor speed increasing at Moore's law rates and disk access time barely changing, the gap between the processor and disk access time is becoming more and more critical. Rapid increase in the speed of microprocessors and the relative slowness of disk access times, being limited by mechanical delays, have caused file servers to fail to deliver the speed they are designed for. Caching can ease congestion by storing the frequently accessed data in main memory, but fitting the entire work set into the memory is not always feasible. Further, it would decelerate other system processes, not to ignore the high cost in achieving such large memory space. To make things worse, we see an almost annual doubling in disk capacity, which is marginally ahead of the decrease in access density or the number of I/Os per second per GB of data. This evinces clearly that, though every generation of disk makes its arm a little faster, each are consigned with a lot more data to handle. To attain good performance on servers, the I/O traffic patterns must hence be known, and the disk must be optimized for such patterns. A plethora of optimization techniques are available including using larger cache, better file system utilization, and varying disk organization techniques. However, without an understanding of the actual access

usage patterns, it is somewhat difficult to make decisions on which to be implemented.

In our thesis, we therefore, first explore the low-level disk access patterns for different workloads for a variety of file systems and use them to get an understanding of the disk usage under those workloads. While there has been a great deal of work done on disk architectures, there has been very little work measuring the actual low-level disk access. Much of the related work has focused on collecting traces at the operating system level, which does not clearly indicate when the disk was actually accessed for a particular request, since the file system behaviors can mask much of the activity at disk. In other words, these traces do not provide the level of disk access detail needed to adequately understand the disk access behavior. These operating system traces are not useful when trying to improve disk storage systems design for file system performance related to disk. Our work extends the previous work done, to also consider the different file systems and examine their effect on the workload.

Next, we discuss the various optimization techniques suggested for improving disk performance. Earlier work has been done to understand the proficiency of each of these under different workloads on different operating systems/servers. In our work, we, in particular, tested ideas which took into consideration not only the frequency of access of different blocks on the disk but also the way they were accessed. We then suggested use of multiple "hot" zones over the disk present at regular intervals instead of using a single "hot" zone present in the center of disk.

## 1.2 File Systems - An Overview

The Linux kernel supports many different types of file systems. The first version of Linux was based on the Minix file system in which, disks were allocated using 1Kb of data blocks. Later, the Extended File System (Ext FS) was introduced, which included several significant extensions but lacked in performance. This was followed by the introduction of EXT2 in 1994 and then EXT3 in 2000. Today there

are numerous file systems available on Linux – JFS, XFS, ReiserFS, LFS etc. In this section we give a brief introduction on the details of the filesystems being characterized in Chapter 2. File systems today are classified under two categories:

(1) Non-Journaling Filesystems

(2) Journaling Filesystems

### 1.2.1 Non-Journaling Filesystems

#### 1.2.1.1 EXT2 Filesystem

EXT2 - the Second Extended Filesystem is native to Linux and is used virtually on every Linux system. This file system was introduced in 1994 and is the most widely used Linux file system. It is quite efficient and robust. EXT2 is an *i-node* based file system; the *i-node* maintains the metadata of the file and points to the actual data blocks. The disk blocks are partitioned into groups where each group includes data blocks and inodes in adjacent tracks which helps in reducing the disk seek time.

### 1.2.2 Journaled Filesystems

Journaling is one of the most critical features required by most of the high performance and highly available servers. Events like power failure or system crash can leave the system in inconsistent state, as quite a lot of data is temporarily allocated in RAM by means of buffer cache and page cache systems. To overcome this, by default, UNIX/Linux performs a filesystem check before each boot. If the filesystem was not properly unmounted, a complete and exhaustive filesystem check is performed on the data structures on disk. This could take lot of time for large disks. To avoid this time-consuming process of looking at the complete filesystem for faults, journaling was introduced. Remounting these filesystems after system crash/failure could be

done in seconds with the help of journaling. Example of these kind of file systems include EXT3, JFS, ReiserFS, XFS etc.

In journaling, each filesystem transaction is logged before and after a request is issued to the system. A filesystem transaction includes operations such as renaming files, copying files, moving files etc. Upon clear completion of the request, it is committed to the filesystem and removed from the journal, just like a database transaction. Thus, in case of a system crash while processing the request, the system could be restored to its normal condition by just replaying the journal and restructuring the data blocks. Linux has four major contenders in the arena, EXT3, JFS, ReiserFS and XFS.

#### 1.2.2.1 EXT3 Filesystem

EXT3 [Twe] is a journaling layer atop the traditional EXT2 filesystem so that recovery of a system in case of crash is much faster than an EXT2 system. EXT3 filesystem differs from others, since unlike JFS or ReiserFS, it can be configured to log both metadata operation as well as data blocks of the files [BC03]. Since logging all the operations have a major effect on the performance of the file system, it is up to the system administrator to decide what has to be logged. EXT3 carries a major advantage that it can be mounted as an EXT2 filesystem as well and is also capable of reading EXT2 filesystem - thus enabling an easy upgrade from its non-journaled version.

#### 1.2.2.2 JFS

IBM's Journaled File System (JFS) [JFS] is the most commonly used file system on enterprise servers. It is designed for high throughput and reliable server environments. It uses extent-based addressing structures, along with clustered block allocation policies. An *extent* is a sequence of contiguous blocks allocated to a file as

a unit and is described by a triple consisting of <*logical offset, length, physical*>. This produces a compact, efficient and scalable structure for mapping logical offsets within files to physical address on the disk. JFS logs are maintained on each filesystem and used to record information about operations on metadata. The log format is set by the filesystem creation utility.

#### 1.2.2.3 ReiserFS

ReiserFS [rei] is an atomic file system. This file system uses fast balanced trees, also known as "dancing trees" for their data blocks layout. ReiserFS introduced "Tail Packing" in their filesystem. Tails are files that are smaller than a logical block, or portions of files which are smaller than a logical block size. This feature gives ReiserFS approximately 5% more disk-space than an equivalent EXT2 system. ReiserFS has an excellent small-file performance because of its capability of incorporating these tails into B+ trees. Though this algorithm makes it a very space efficient filesystem, ReiserFS cannot recover from any media faults. For example, a bad-sector on a drive could cause a complete data loss, while EXT2/EXT3 can "force read" past the bad sector for recovery.

## 1.3 RAID Systems

Redundant Arrays of Inexpensive Disks (RAID) is a stack of two or more disk drives which operates as a single unit. In general, these drives could be any storage system, such as magnetic hard drives, magnetic tapes or any optical storage. Under conditions where speed is an issue, SCSI drives could be used. RAID systems offers lower cost per byte, and have built in controller logic which can perform both error detection and correction, thus being more reliable. The key to protecting data is to store parity data, so that if a drive fails, the remaining drives will have data, that was also on the failed disk.

### 1.3.1 RAID Levels

RAID Levels indicate how the hard drives in a system are working together. The most commonly used RAID levels are RAID 0,1,3 and 5. In our thesis, we use RAID 5 for our experiments.

#### 1.3.1.1 RAID Level 5 - Striped Parity

RAID 5 carries good parity performance by reducing the choking on parity disk due to multiple write requests being queued on it. RAID 5 stripes the parity on the different disk, thus removing this bottleneck. This scheme also allows simultaneous writes if the writes are to different stripes, and there are no common clusters between the writes.

## 1.4 Understanding of Disk Time

The time taken by a disk to respond to a particular request is composed of two parts - *access time* and *transfer time*. Access time refers to the time taken by the disk head to position itself over the correct sector and comprises the delay in moving the arm to the correct track and the time spent waiting for the requested sector to rotate under the head. Transfer time is the actual time needed to read the requested data after the head is in position. This implies that transfer time is the actual time of performing read/write operation, and it comes with access time as overhead. Thus, requesting large sequential chunks of data would be beneficial, as that would reduce the disk access time, and hence, the overall response time. However, if the data read sequentially is not accessed again, there would be no need to cache it in RAM. If this were the case under the different workloads, memory of around a mega-byte would be sufficient for a good performance [GG97]. But this does not happen. Most of the sequential read of data is referenced again, either in part or in full. Our experiments in Chapter 3 would assist this statement. In our experiments, we found that around

2000–3000 blocks on a read-dominated server, and 20,000–30,000 blocks on an equal read/write server were referenced again within an interval of 15 minutes. This figure turns out to be 30–40% of the total number of blocks accessed during that period. Keeping this fact of revisiting of blocks in mind, several algorithms based on relocation of frequently accessed blocks have been designed.

## 1.5 Outline

This thesis has been organized into 4 chapters. The first chapter gives the introduction and motivation for this work. It also briefly discusses the various file systems and provides an understanding of the response times. Chapter 2 discusses our approach in tracing disk at the low-level under Linux kernel. Chapter 2 also provides our results and observations of the effects of the workloads. Chapter 3 discusses the different algorithms being used until now and introduces our algorithm. Our results on single disk as well as on RAID systems are also showed. Finally, in Chapter 4, we conclude with our ideas for future work.

# Chapter 2

# Disk Tracing in Linux Kernel

## 2.1 Introduction

With the rapid widening of gap between processor performance and disk access time, it becomes increasingly important to understand the characteristics of I/O patterns on disk. Because the disk technology has not kept pace with the improvement of other components, it is the file system which needs to be efficient in its management of disk resources. To do this, a file system developer must understand the access pattern of the data on disk for different workloads. Traces of the disk subsystem can provide the maximum information on how the disk is treated by a particular file system. To improve the disk access times, one should be familiar with how the data is delivered to the storage system by the file system. These traces are important for many purposes other than the algorithms. These traces are difficult to obtain, and not much work has been published in this area. Much of the related work has collected traces at the operating system level, which does not clearly indicate when the disk was accessed for a particular request, since the file system behaviors can mask much of the activity at disk.

In this part of thesis, we show an updated analysis of I/O system disk behavior based on a variety of workloads, both real and simulated. We also examine whether the choice of file system has an effect on low-level I/O access.

## 2.2 Related Work

There has been significant work done on the analysis of I/O system performance using traces collected from file systems [BHK+91, KRK92, Vog99, OCH+85, RLA00, Mil91]. One of the earliest studies was the BSD study and its follow up Sprite study [BHK+91, OCH+85]. The BSD paper has provided much of the foundation for latter file system research. Roselli et. al. re-examined the conclusions from the BSD study with a set of traces collected from several HP/UX servers and a collection of Windows NT clients [RLA00]. There have also been several studies of distributed file system access traces [ELMS03, GNA+97]. The procedure used in these and similar studies was to capture the traces at a high level - either at the file system level, or network level. These traces do not illustrate when the access was actually made to the disk, if at all made in the first place. The UNIX buffer system stores the data temporarily in cache. The data could hence have been hived away in cache, and later deleted from cache memory itself - thus a request for that file need not have been made to the disk.

The most relevant work at the disk level was that done by Ruemmler and Wilkes at HP Laboratories in 1993 [RW92, RW93]. Their work generated traces from the disk level access collected on HP-UX system with BSD Fast File System mounted. Their results indicated that a small non-volatile cache at each disk allowed writes to be serviced much faster than any regular disk. Similar work was done by researchers at IBM where low-level traces were collected on a variety of systems including Windows NT PCs and IBM AIX and HP-UX servers [HS03]. As in the HP study, they found a high degree of burstiness in the access pattern, as well as a write-dominated workload. Idleness in the storage system also suggested opportunities for background optimizations as discussed later in the thesis. While the IBM study did examine different file systems, it was difficult to determine the effect of the file system on I/O accesses since the workloads also changed. In our work, we intend to further

examine the effect of the file system by tracing the same workload on different file systems [NC04].

## 2.3 Trace Collection

We sought to extend the previous work by examining the effect of the file system on I/O access patterns. Since the system under study was a Linux server, we traced four popular file systems, namely EXT2, EXT3, JFS and ReiserFS. EXT2 is the most commonly used file system in client Linux systems. EXT3, JFS and ReiserFS are all journaling file systems, in that, fast restart is enabled through file system metadata logging techniques [TT02, Bes00]. We examined these file systems under three different workloads – an actual NFS email server workload, a synthetic CIFS file server workload and a database workload.

To capture the disk I/Os at the lowest level, we introduced a thread into the IDE driver of Linux kernel 2.6.0. The code was designed in such a way, that only one particular disk could be tested. The differentiation was done based on the drive numbers of the disk. The thread was transparent to user, and did not add much burden to the system's performance. This thread would record all the requests sent to the disk, along with many other details. Each record collected the following details:

(1) Time of Request

(2) Device Number

(3) Sector number to which the request was made

(4) Request Size

(5) Action (Read/Write)

The collected trace was written at regular intervals to a completely different disk, to avoid this write as being recorded as another request on the test disk. The average

size of each of such write was approximately 600KB.

To trace the NFS email server, we used the CAMPUS trace from a high level network file access study conducted on Harvard University's email server [ELMS03]. Using this trace, we reconstructed the file tree which would have existed on the CAMPUS server at Harvard University at the time they collected the trace. Similar file tree reconstruction techniques are described in [Bla92, ELMS03]. While this is not an exact replica of the CAMPUS server, it replicated enough information so that we can replay the trace and collect meaningful data. This tree was then replicated on four different servers, each built with one of the targeted file systems. Upon generation of the file system tree, an NFS client was programmed to generate the requests to the server as if a real request for I/O was made to the server. For this purpose, we replayed a week's worth of Harvard's trace. To make our framework more precise, we sent the request to the server with the same time interval, as the trace file stated. On the server side, the thread was initiated. The CAMPUS trace also included userid, groupid, and IP information – these were ignored, since they were not relevant to our experiment.

The second workload is the synthetic Netbench [Ver01] benchmark which exercises CIFS file servers. We used the Disk Mix Test Suite to generate the synthesized emulation of network file server access from Windows client in an office environment. While Netbench may not be an ideal real-world trace, it is based on real office usage patterns. More importantly, it does exhibit more write-oriented behavior as opposed to the CAMPUS NFS server.

The final workload is the OSDL's Database Test Suite (OSDL-DBT2) [OSD03], inspired from the TPC-C benchmark [Tra92]. It is an online transactional processing test, which simulates an inventory control database with several workers accessing the database for purposes such as viewing, updating etc. The test was conducted for 50 warehouses under single connection for 18,000 seconds. The database we used for

data management in this test was PostgreSQL 7.4.

## 2.4 Traced System

The systems which acted as NFS clients were Pentium 4, 2.4 GHz computers with 512 MB of RAM each, while the server configurations were Pentium III, 800 MHz computers with 512 MB RAM. Each file system was mounted on one individual server and a unique client was connected to it to perform the experiment.

For the Netbench [Ver01] workload we used a Client and Controller Windows machine to run the Netbench software. Using Samba, these Windows systems were connected to the test Linux server and separate servers were again used with different file systems mounted and traces were collected.

For the OSDL-DBT2 [OSD03] workload, we used the same Pentium 4 configuration as servers for testing with fresh mount of the file system. With each run, a fresh installation of PostgreSQL was done, a new disk was mounted and a new database was created.

## 2.5 Analysis

In this section, we analyze the traces collected in our experiments. Our analysis is based on the following categories:

(1) Read/Write frequency

(2) Inter-arrival Time Distribution

(3) Request Size Distribution

(4) Burstiness

### 2.5.1 Read/Write Frequency

Read/Write frequency is the percentage of read and write requests which were made on the disk. We use this parameter to define the kind of workload we are using. We obtained this ratio for each file system under each workload. The exact figures for the read/write frequency can be seen in Table 2.1–2.3.

Since the CAMPUS server is predominantly an email server, it is natural to expect it to be read-dominated. The Harvard data indicated the same, as they found three times as many reads as writes. At the disk level, we found however, the ratio of reads to writes is significantly higher due to the extra metadata activity that is not present in the NFS trace. Our observation showed that the CAMPUS email sever [ELMS03] was read-dominated, Netbench [Ver01] write dominated, and OSDL Database Workload [OSD03] had equal read/write ratio. All four file systems followed the similar pattern. It is interesting to see that the type of workload can have such a dramatic effect on the read/write frequency. Particularly, the fact that the email server is heavily read-oriented should lead storage system designers to optimize reads even at the expense of writes. Even though, the Netbench numbers seem to indicate that file servers should be write-optimized, this needs to be taken with a grain of salt. The Netbench data set is only 22MB in size which can easily fit in cache, thus explaining why there are so few reads. In the context of the traces we are collecting, however, the Netbench data is useful since it serves as an example of a write dominated workload. Finally, the database workload is a good example of a workload that is evenly divided between reads and writes.

In terms of raw numbers, there are some points that stand out. With the heavily write-oriented Netbench workload, we see that the journaling file systems, EXT3, JFS and ReiserFS, have nearly three times as many writes than EXT2. These extra writes are due to the writes to the journal. However, one may notice that the database workload does not show a similar increase in writes. The reason is that the database

workload does only file writes without any file system/metadata operations such as deleting a file, renaming a file, moving a file, etc. while on the other hand, the Netbench workload does a lot of these operations. File writes are not journaled, but file system modifications are journaled - thus the difference in behavior.

| | **EXT2** | **EXT3** | **JFS** | **ReiserFS** |
|---|---|---|---|---|
| **Read** | 8,998,227 | 8,464,299 | 7,215,712 | 7,010,054 |
| **Write** | 246,579 | 337,898 | 182,263 | 154,981 |
| **Total** | 9,224,806 | 8,802,197 | 7,397,975 | 7,165,035 |
| **% Read** | 97.33 | 96.16 | 97.54 | 97.83 |
| **% Write** | 2.67 | 3.84 | 2.46 | 2.16 |

Table 2.1: Read/Write Frequency for Email Workload (Read Dominated)

| | **EXT2** | **EXT3** | **JFS** | **ReiserFS** |
|---|---|---|---|---|
| **Read** | 30 | 48 | 21 | 25 |
| **Write** | 16,720 | 45,991 | 46,193 | 47,081 |
| **Total** | 16,750 | 46,039 | 46,214 | 47,106 |
| **% Read** | 0.18 | 0.10 | 0.05 | 0.06 |
| **% Write** | 99.82 | 99.90 | 99.95 | 99.94 |

Table 2.2: Read/Write Frequency for Netbench Workload (Write Dominated)

| | **EXT2** | **EXT3** | **JFS** | **ReiserFS** |
|---|---|---|---|---|
| **Read** | 2,940,367 | 2,893,065 | 2,331,008 | 1,410,985 |
| **Write** | 2,156,496 | 2,125,214 | 1,712,358 | 1,074,663 |
| **Total** | 5,096,863 | 5,018,279 | 4,043,366 | 2,485,648 |
| **% Read** | 57.68 | 57.65 | 57.65 | 56.76 |
| **% Write** | 42.38 | 42.35 | 42.35 | 43.23 |

Table 2.3: Read/Write Frequency for Database Workload (Equal Read/Write)

### 2.5.2 Inter-arrival Time

Inter-arrival time is defined as the time between successive disk access requests. It is a useful measure to help identify the load on the disk system. In particular, the average inter-arrival time determines the required service time from the disk. The

inter-arrival time can also be used to develop workload models that can drive queuing studies. Inter-arrival Time distribution for the same test for different file systems showed different patterns. Figures below show the comparison of the different file systems (EXT2, EXT3, JFS and ReiserFS) under various workloads with respect to the inter-arrival time in each case.

#### 2.5.2.1 Email Server (Read Dominated)

It is clear from the distribution graph shown in Figure 2.1 that most of the requests are on average less than a second apart, with the majority of them less than about 400ms apart for EXT3 and 600ms for EXT2. EXT3 showed a completely different characteristics from its counterpart JFS and ReiserFS.

A steep rise was observed in the count of inter-arrival requests between 0.2 and 0.4 seconds for EXT3, while EXT2 and JFS showed spikes near 0.6 seconds and 1 second. Figure 2.2 shows the distribution graph plotted for the read request alone, which showed similar characteristics to that of the overall inter-arrival cumulative distribution graph.

Since nearly half of the inter-arrival times for the read requests are very short, we magnified the graph for the region between 0 and 0.15 seconds (Figure 2.3). As can be seen, nearly half of the inter-arrival times are at 0.01 seconds. These short interval times are due to back to back NFS requests caused by a large NFS read broken into smaller NFS requests due to protocol limitations. To know the inter-arrival distribution of the write requests, we plotted the graph for write requests also (Figure 2.4). All four file systems indicated almost the same behavior with EXT3 showing slightly more frequent access.

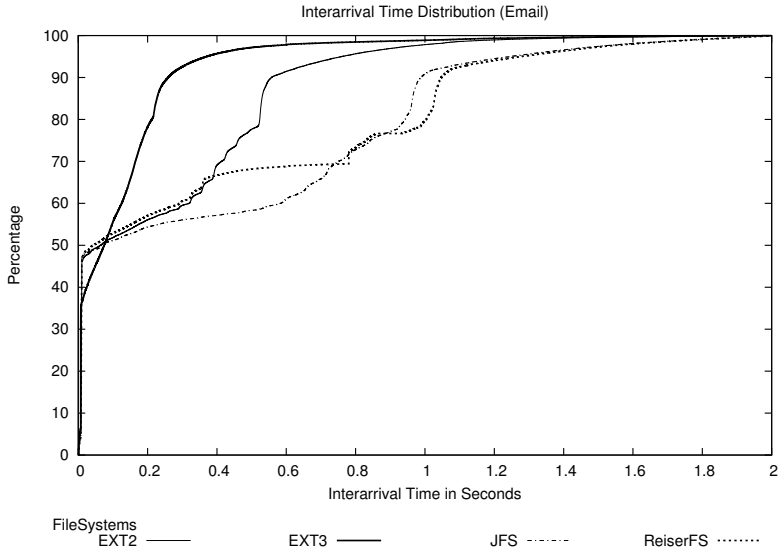


Figure 2.1: Interarrival Time Distribution for Email Server

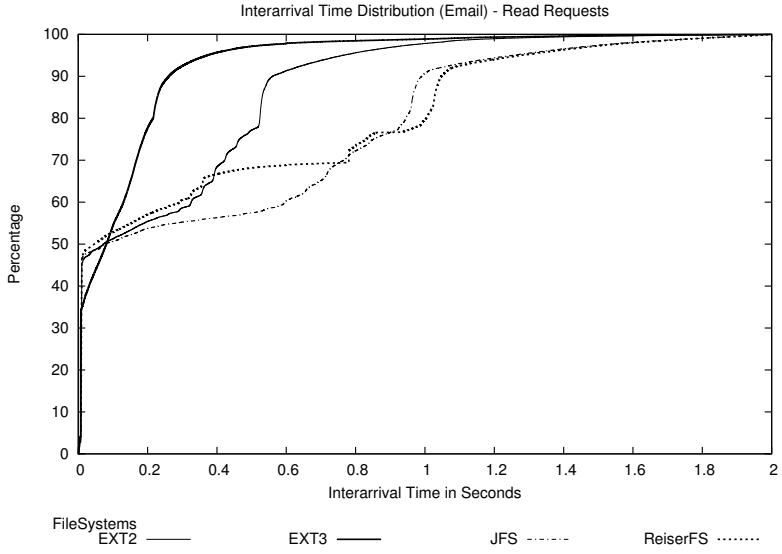


Figure 2.2: Interarrival Time Distribution for Email Server - Read Requests

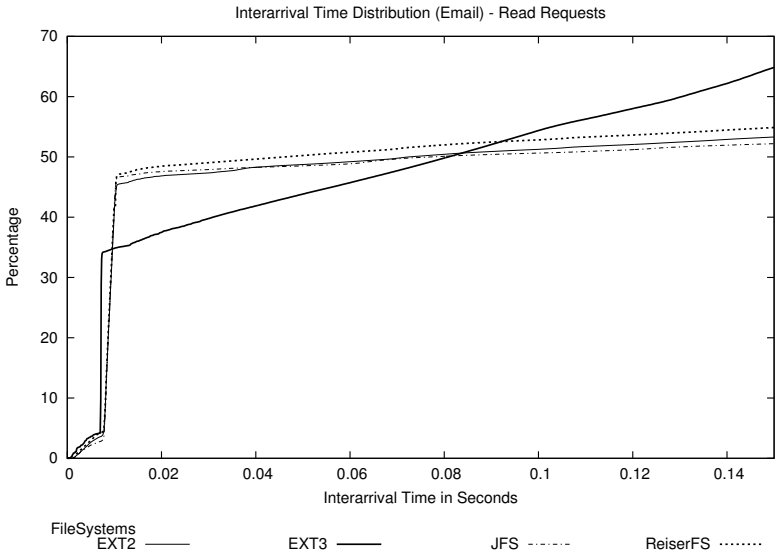


Figure 2.3: Interarrival Time Distribution for Email Server - Read Requests

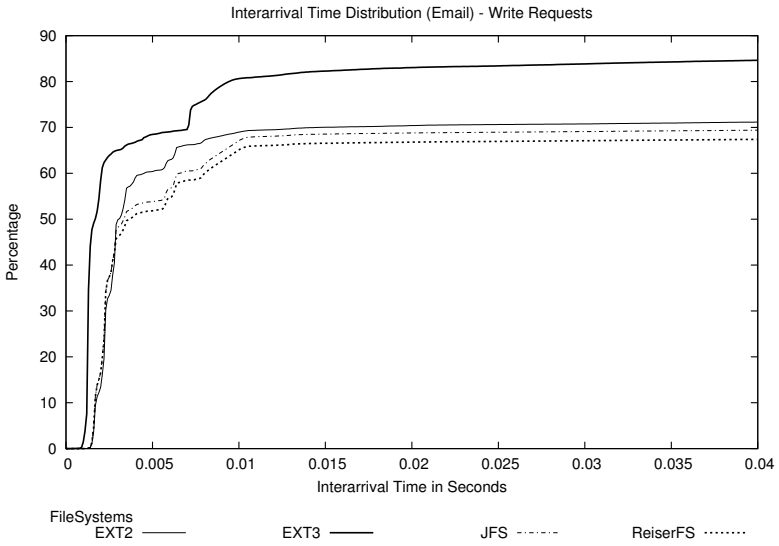


Figure 2.4: Interarrival Time Distribution for Email Server - Write Requests

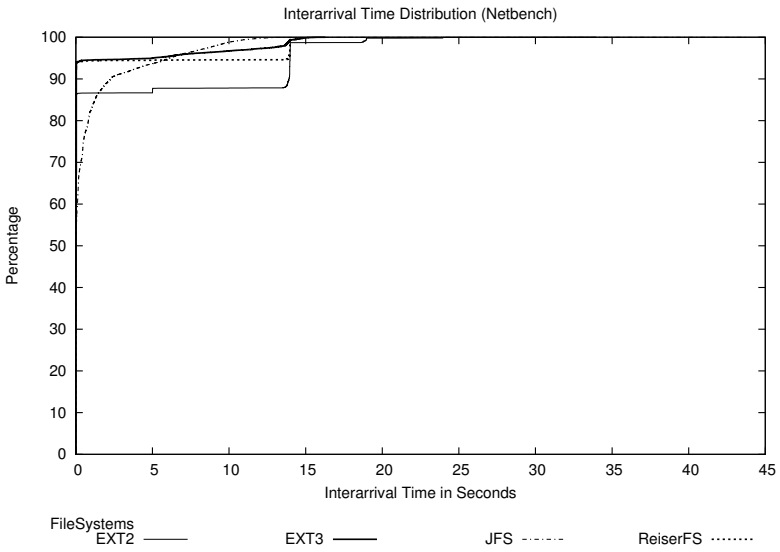


Figure 2.5: Interarrival Time Distribution for Netbench

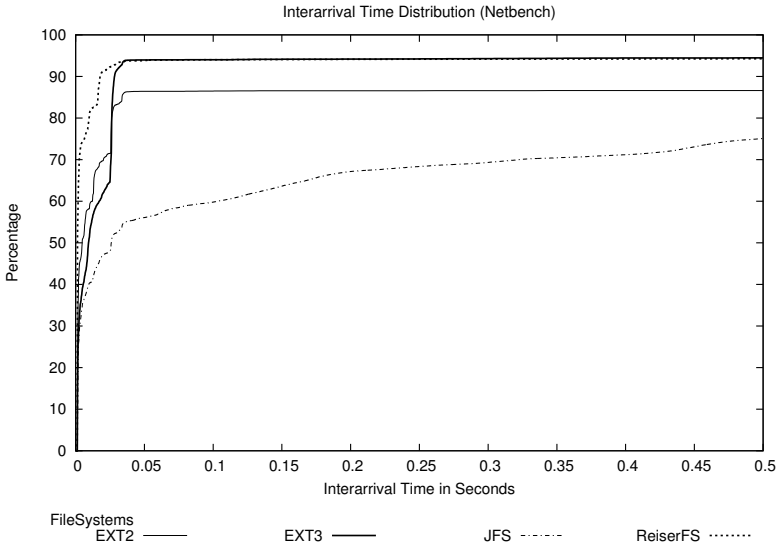


Figure 2.6: Interarrival Time Distribution for Netbench

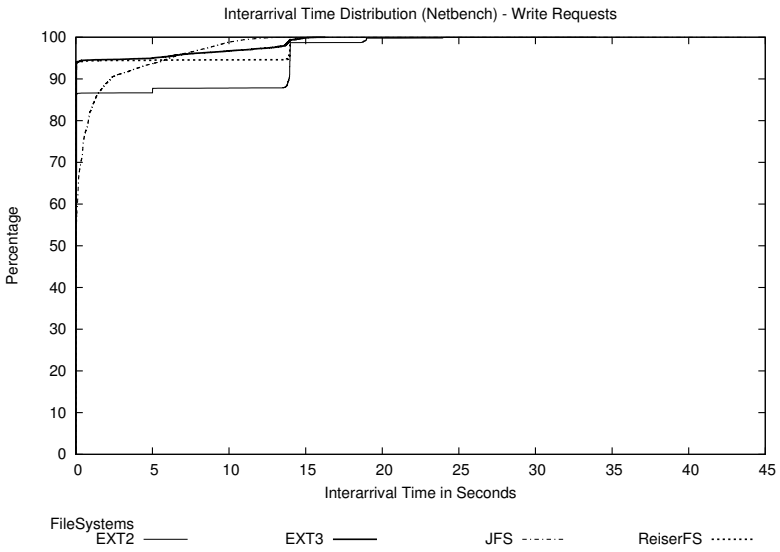


Figure 2.7: Interarrival Time Distribution for Netbench - Write Requests

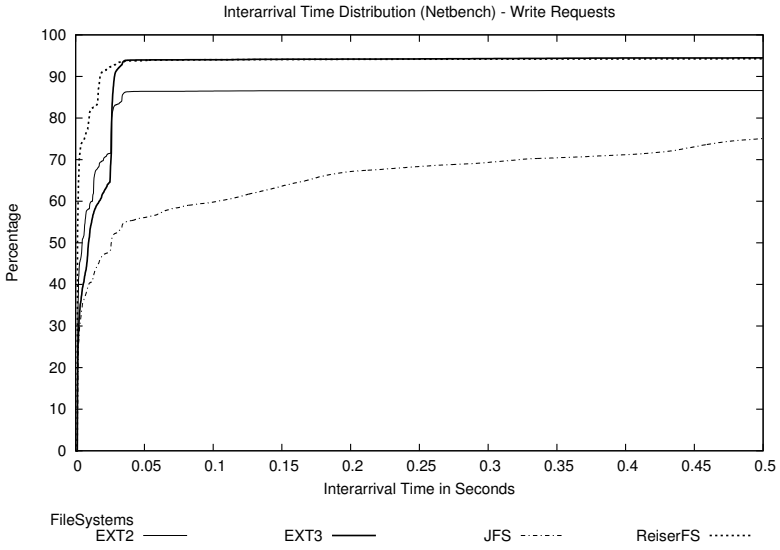


Figure 2.8: Interarrival Time Distribution for Netbench - Write Requests

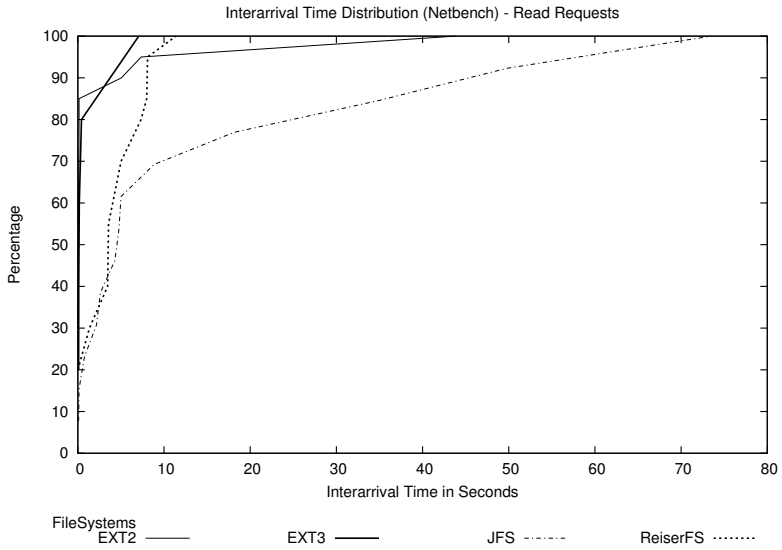


Figure 2.9: Interarrival Time Distribution for Netbench - Read Requests

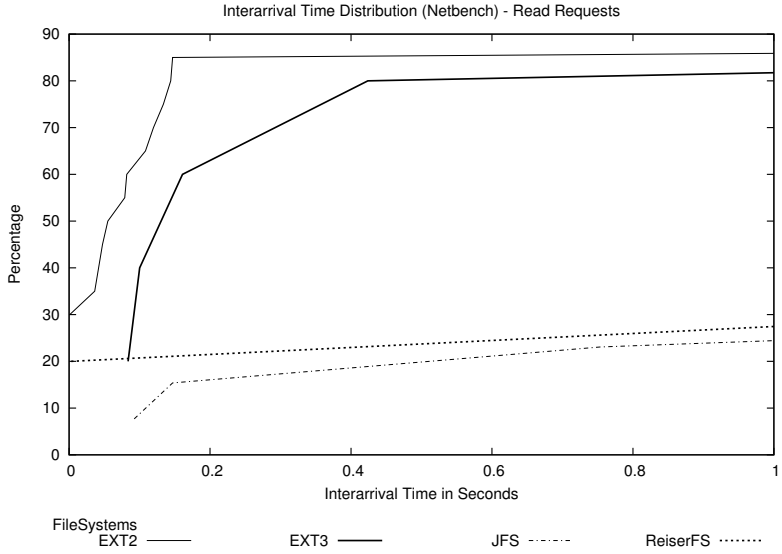


Figure 2.10: Interarrival Time Distribution for Netbench - Read Requests

#### 2.5.2.2 Netbench (Write Dominated)

On Netbench, EXT2 and EXT3 showed almost the same pattern, with EXT3 having slightly more frequent accesses (Figures 2.5 and 2.6).

Since Netbench is write dominated, the write access distribution is very similar to the overall interarrival time distribution (Figures 2.7 and 2.8). Looking at Figure 2.6, once can see that EXT2 and EXT3 have interarrival times that are much more shorter than JFS. Nearly 90% of the intervals with EXT3 and ReiserFS are less than 0.02 seconds. JFS tends to delay writes, and thus eliminate some writes through the cache, thus accounting for the difference. It would seem that because of this behavior, JFS based storage system would have longer periods of idle time. Idle time can be used by disk systems to run background optimization schemes, and this graph shows that at least for the write-dominated data, the choice of the file system can influence the length of idle time available. Figure 2.9 and 2.10 shows the same type of behavior for reads but on a different scale. Note that Netbench has so few reads, this data is not as meaningful.

#### 2.5.2.3 Database Workload (Equal Read/Write Ratio)

Figure 2.11 and 2.12 shows the interarrival time distribution for the database workload. The most striking feature in this workload is that the interarrival times are much more frequent. Interestingly, the JFS and ReiserFS filesystems shows much more frequent access than EXT3. This behavior is the complete opposite of what was observed for the read-dominated and write-dominated workloads. Nearly 50% of the interarrivals are less than 3ms for both JFS and ReiserFS, while for EXT3, the 50% mark is reached only at 45ms.

To evaluate this further, we looked at reads and writes separately as shown in Figures 2.13–2.16. It can be seen that there is very little difference between the file system for reads. However for writes, there is a marked difference. For JFS, about

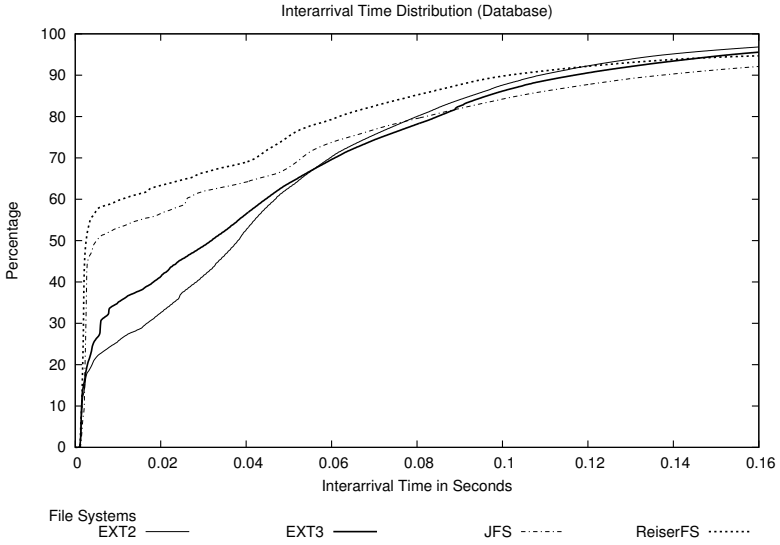


Figure 2.11: Interarrival Time Distribution for Database Workload

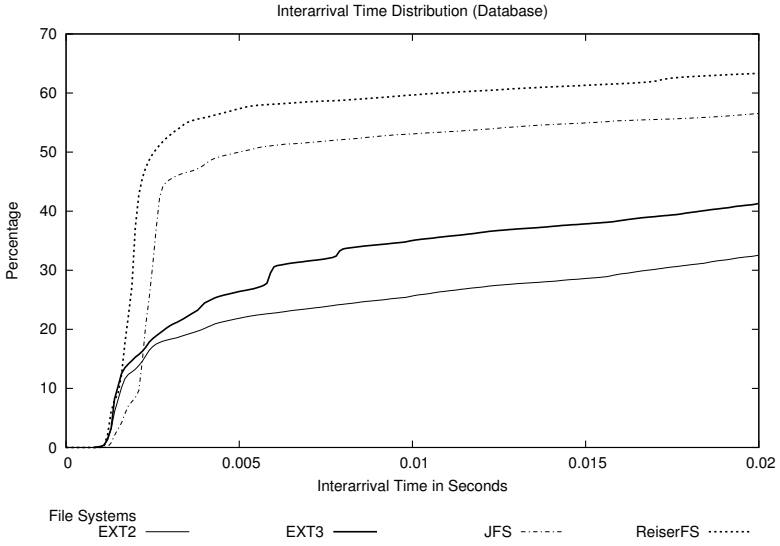


Figure 2.12: Interarrival Time Distribution for Database Workload

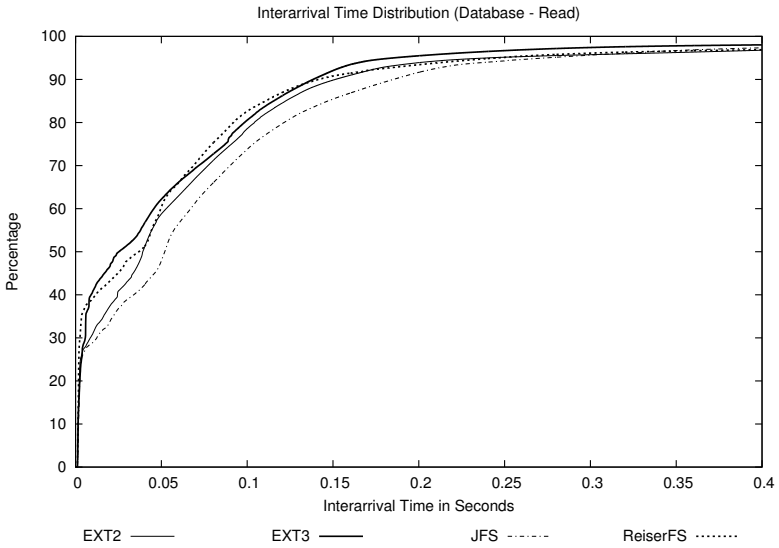


Figure 2.13: Interarrival Time Distribution for Database Workload - Read Requests

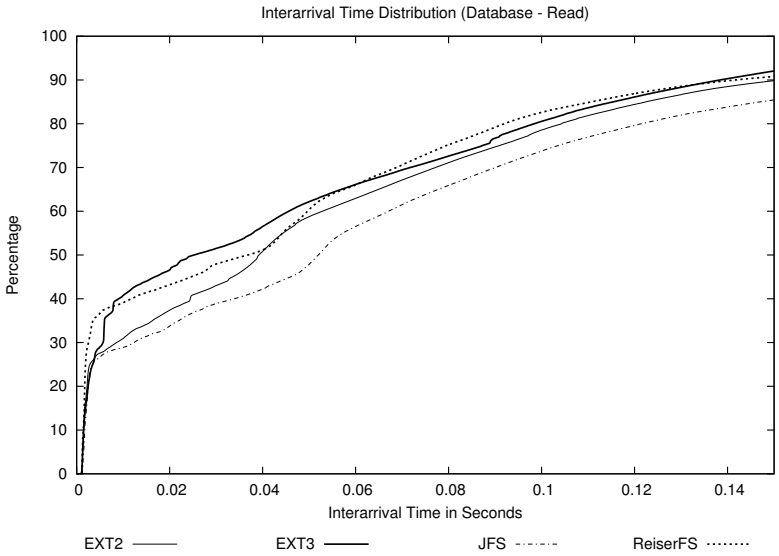


Figure 2.14: Interarrival Time Distribution for Database Workload - Read Requests

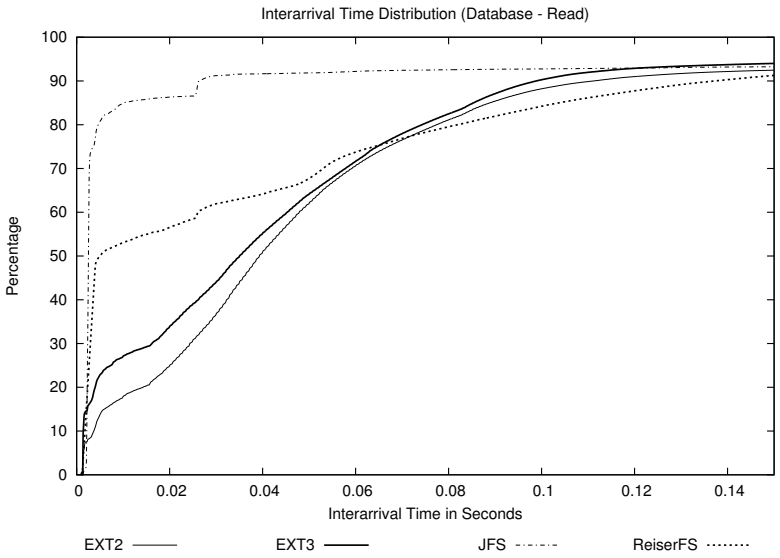


Figure 2.15: Interarrival Time Distribution for Database Workload - Write Requests

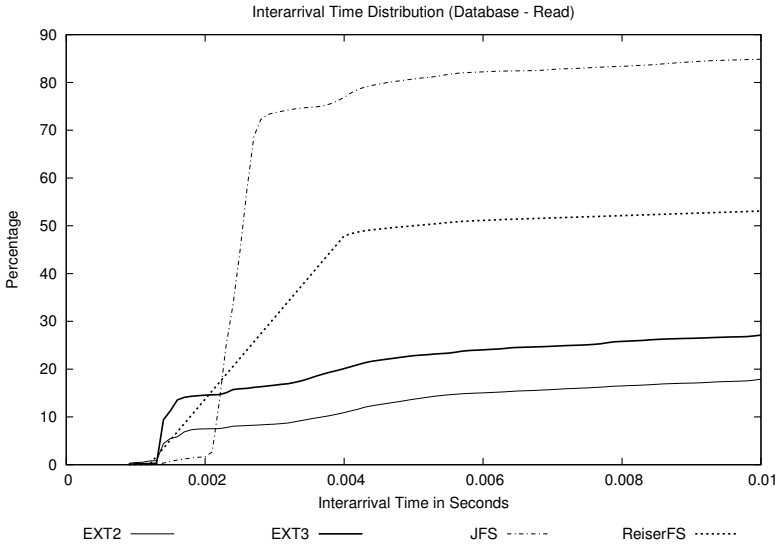


Figure 2.16: Interarrival Time Distribution for Database Workload - Write Requests

80% of the intervals are less than 10ms, whereas for EXT3, the 80% mark is reached only at 100ms. It is clear that JFS does not do a good job of grouping writes, when reads are mixed with the writes. ReiserFS has almost 50% of requests within time interval of 5ms.

### 2.5.3 Request Size Distribution

In this section, we examine the request size distribution, a measure of the number of blocks in each request sent to the disk. We discuss the request size in each file system under the different workloads.

#### 2.5.3.1 Email Server (Read Dominated)

Request size distribution for the email server is shown in Figures 2.17–2.19. The read dominated email server mostly had large reads. There was a peak seen at 64Kbytes. All the four file systems followed the same pattern of block distribution.

The read and write requests are separately shown in Figure 2.18 and 2.19 respectively. While the read behavior is the same for all file systems, for writes, however we notice a difference. Most write requests in EXT2 and JFS are small in size when compared to EXT3. Even ReiserFS had almost 95% of its request with block size less than 8Kbytes. Please note the change in scale on y-axis. It can also be observed that JFS did not show large writes at all. JFS for reliability reasons do not coalesce writes, thus explaining the lack of large writes.

#### 2.5.3.2 Netbench (Write Dominated)

Netbench being a write dominated workload, read accesses on disk did not play much role in its request size distribution graph (Figures 2.20 and 2.21). All the read requests on disk under the Netbench workload were 4Kb in size, the default block size. All four file systems showed a small spike around 64Kbytes, as also observed in

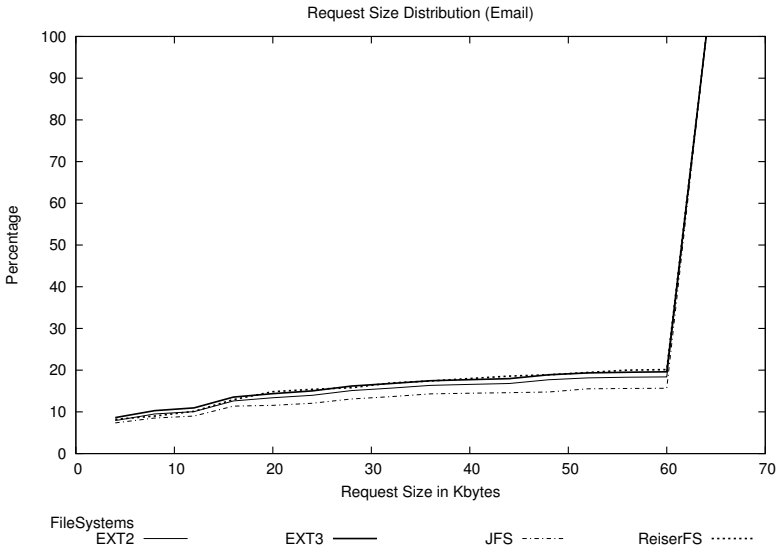


Figure 2.17: Request Size Distribution for Email Server

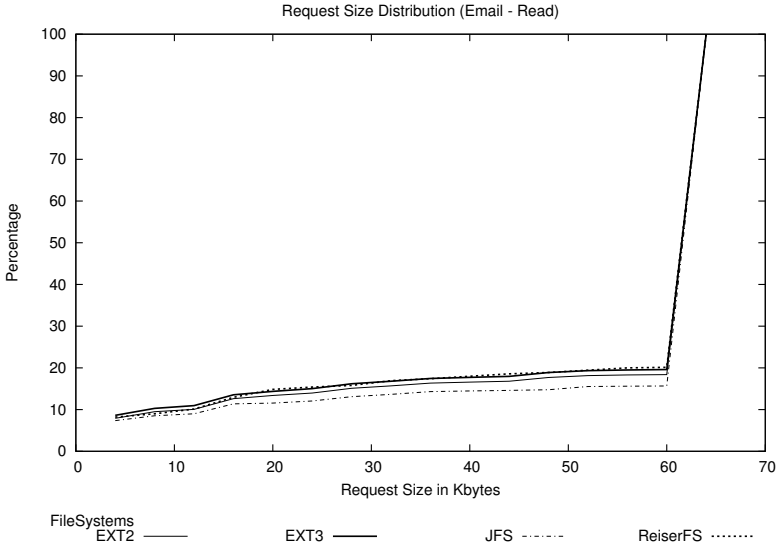


Figure 2.18: Request Size Distribution for Email Server - Read Requests

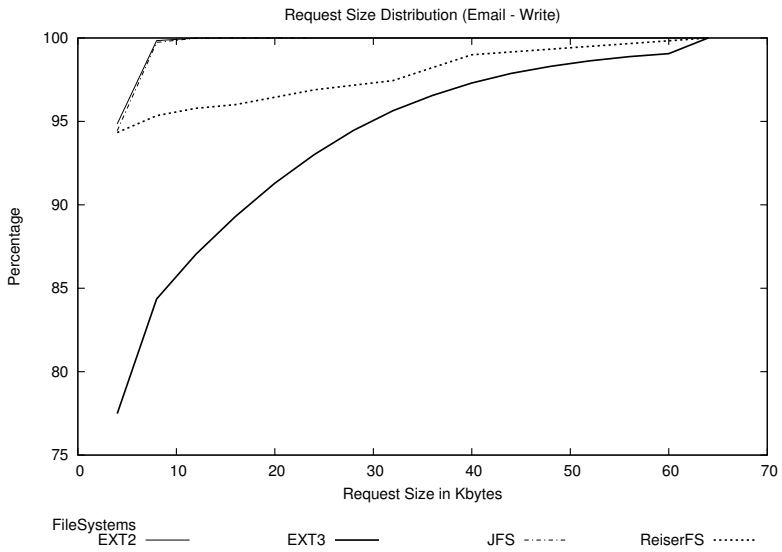


Figure 2.19: Request Size Distribution for Email Server - Write Requests

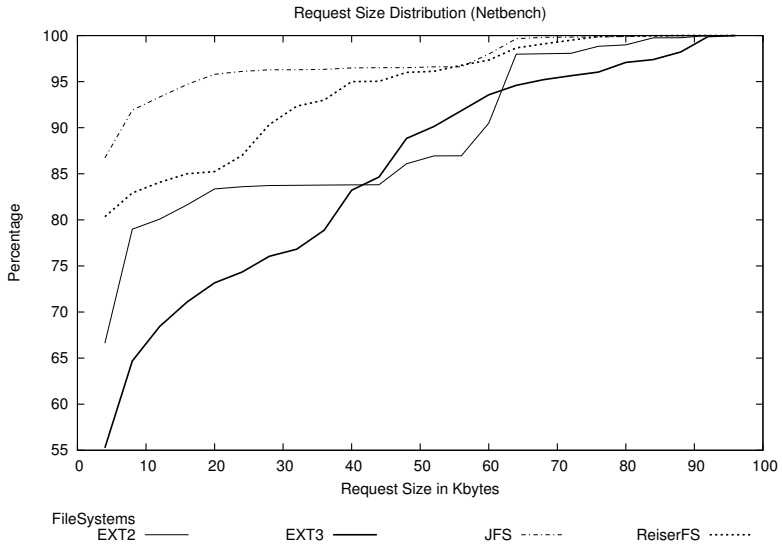


Figure 2.20: Request Size Distribution for Netbench

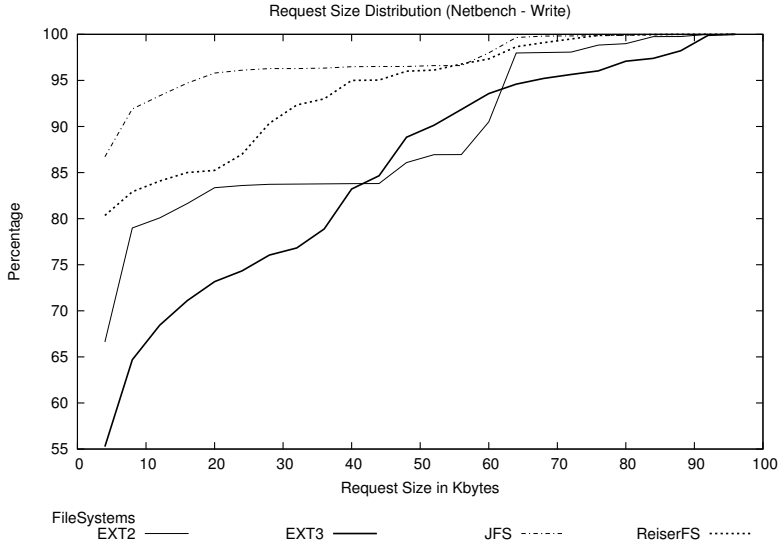


Figure 2.21: Request Size Distribution for Netbench - Write Requests

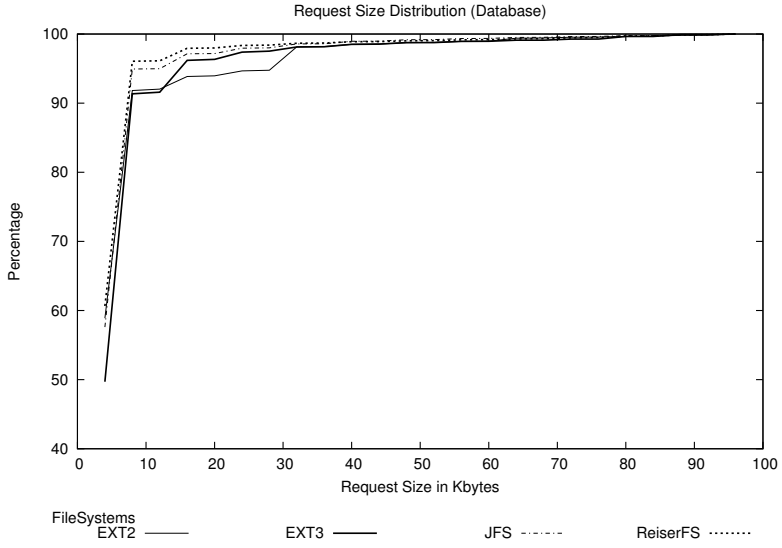


Figure 2.22: Request Size Distribution for Database Workload

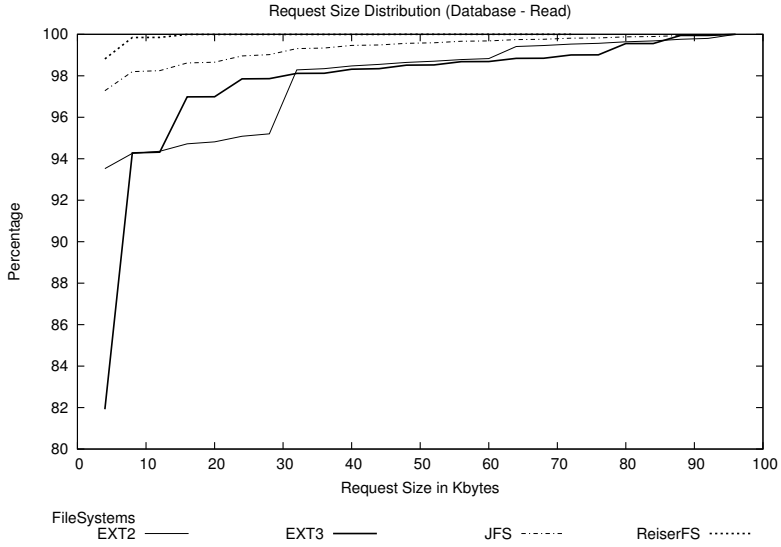


Figure 2.23: Request Size Distribution for Database Workload - Read Requests

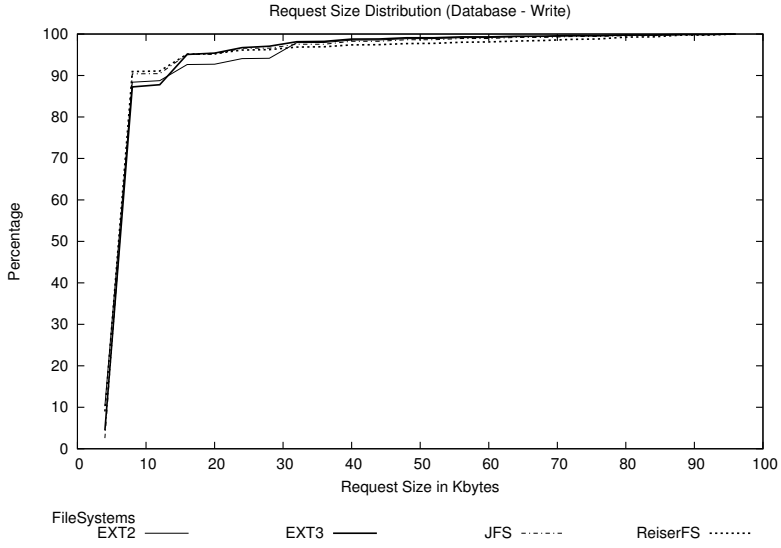


Figure 2.24: Request Size Distribution for Database Workload - Write Requests

the read dominated email server. Both JFS and ReiserFS had majority – over 80% of its requests of block size 4Kbytes.

#### 2.5.3.3 Database Workload (Equal Read/Write Ratio)

Figures 2.22–2.24 show the request size distribution for the database workload. As can be seen, the database workload issues very small requests. Almost all its requests were 8Kbytes in size. This behavior did not show much variation over the different file systems. EXT3 showed slightly fewer small read requests. ReiserFS had almost all of its request under the block size of 8Kbytes. The write pattern was dominated by small writes. Under this workload again, there was hardly any differentiation visible in the four different file systems distribution.

### 2.5.4 Burstiness

We define burstiness as a series of requests made on the disk, which lie within the time interval of 10ms of the previous request sent to the disk. It is a measure of how many accesses are in groups and provides a rough measure of how much idle time is available. The graphs below show the cumulative distribution function where the x-axis is the number of requests in the burst.

#### 2.5.4.1 Email Server (Read Dominated)

Under the read dominated workload, the three file systems did not show any variation from one another. No more than 3 requests occurred in any burst.

#### 2.5.4.2 Netbench (Write Dominated)

Under the Netbench workload, the three file systems showed significant differences in the number of requests coming in bursts (Figure 2.26). JFS had around 80% of its bursts as singletons, i.e. single request bursts, whereas for EXT3, the bursts were

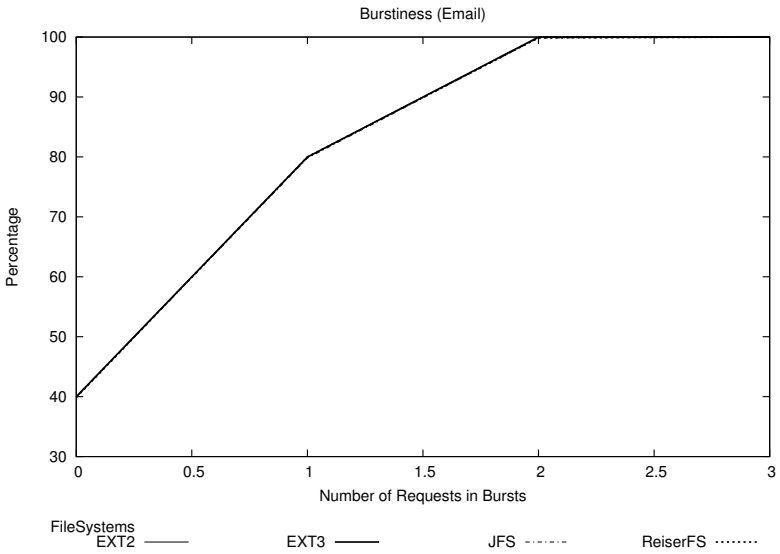


Figure 2.25: Burstiness for Email Server

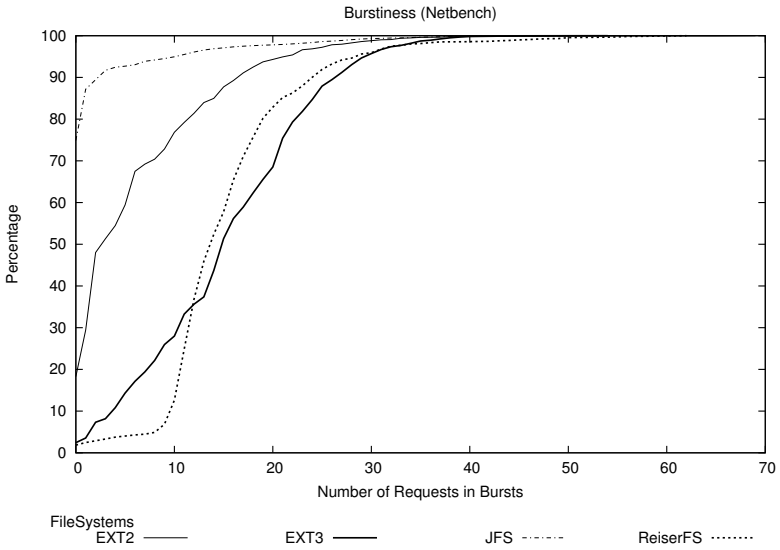


Figure 2.26: Burstiness for Netbench Workload

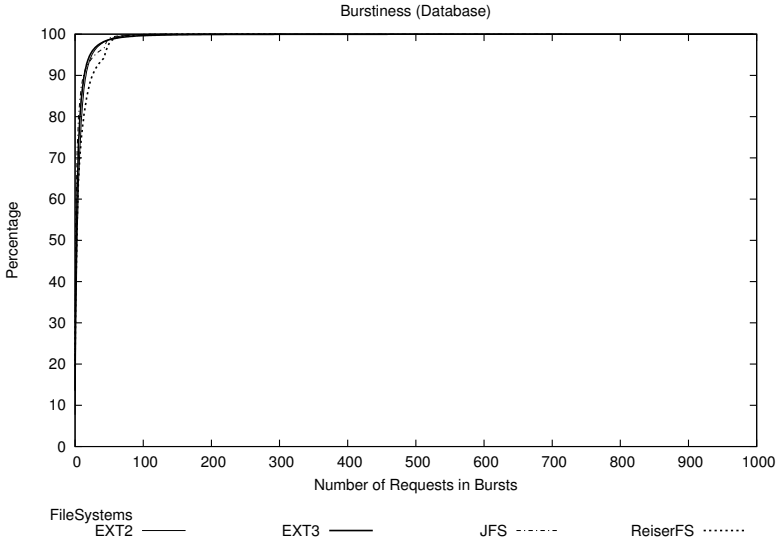


Figure 2.27: Burstiness for Database Workload

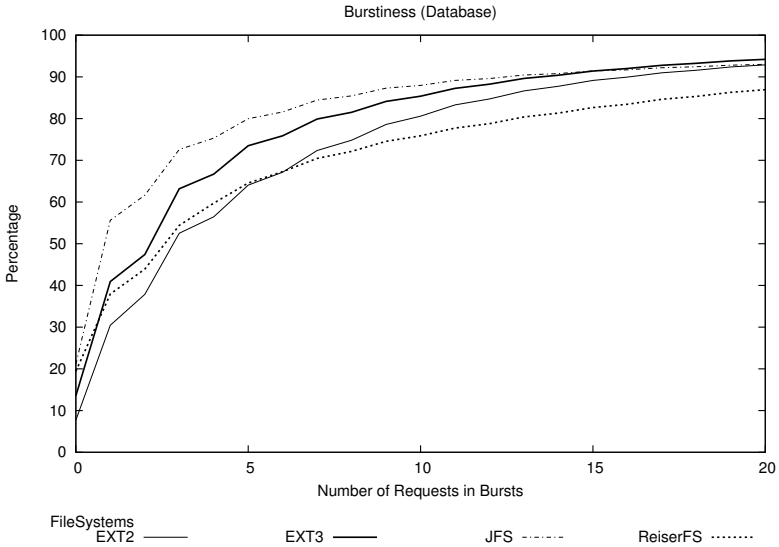


Figure 2.28: Burstiness for Database Workload

typically much larger. Again this behavior is because JFS is quite conservative in terms of coalescing writes to preserve reliability.

#### 2.5.4.3 Database Workload (Equal Read/Write Ratio)

Under the database workload, the four file systems showed similar patterns of very large amount of small bursts (Figure 2.27). The graph was hence magnified for better understanding (Figure 2.28). It can be seen that even in this workload, JFS seems to have shorter bursts than any other file system.

## 2.6 Summary

In this chapter, we have presented new studies of disk I/O traffic under different workloads and different file systems. These findings provide insight to storage and file system designers and moreover highlight the importance of file system choice in designing a storage system. System administrators can also analyze their workload and file system and pick a storage system that may match their particular workload and file system needs.

The key findings are:

- Email servers are heavily read dominated ($> 95\%$ reads). This may indicate that in email servers, large caches either in the server or at the disk are warranted.
- File Servers are very write dominated. Write dominated systems may be able to take advantage of log-structured file systems or disks [DO89, WAP99].
- The file system has a very minor effect on read-dominated workloads
- For write-dominated workloads, journaling file systems can cause an increase in small writes. This may lead an administrator to add NOVRAM to coalesce writes or not use RAID5 to address small write problem with journaling disks.

- In an email server workload, there is not much burstiness, and what little there is, consists mostly of 2-3 request bursts. However, for a write dominated file serving workload, EXT2 and EXT3 show significant burstiness as JFS. The presence of burstiness may indicate more available idle time to do any background work on disk — for example disk defragmentation, log cleaning etc.

# Chapter 3

# Optimization using Block Reorganization Techniques

## 3.1 Introduction

With disk I/O being the choke point in the computer systems, several efforts have been introduced to reduce the disk seek time. Processor performance increases at more than 50% per year, while disk access time improves at only about 10%. This gap between the processor performance and disk storage has been widening more and more every year. The lack in the improvement in response times of the disk have caused file servers to fail to perform at the speed they could deliver.

There have been numerous studies on rearranging data blocks to reduce the disk access time. Data access on disks is often skewed, and thus block re-arrangement techniques could be very advantageous as they would place the most frequently accessed blocks together, and hence reduce seek time of the disk. However, read-response time varies largely according to the rotational latency of disk. Also, inertia and head-settling time play a role. Because of inertia and high head settling time, not much difference would be observed between short seek and long seek times particularly on newer disks. Thus, reducing the disk seek time alone would not be very helpful with the new trends in disk manufacture. We need a method which can reduce the number of physical I/Os along with the seek time. Various heuristics have been developed

to layout data on disk so that items that are expected to be used together are located close to one another [AS95, BEBW98, RW91]. The shortcomings in these a priori techniques is that they are based on static information such as name-space relationships of files, which may not reflect the actual reference behavior [Hsu03]. Furthermore, files become fragmented over time. The blocks belonging to individual files can be gathered and laid out contiguously in a process known as defragmentation [Don88]. But defragmentation does not handle inter-file access patterns and its effectiveness is limited by the file size which tends to be small. Defragmentation also holds on to an assumption that the blocks belonging to the same file tend to be accessed together which may not be true always, for example, large files or database tables.

Several strategies of placing the frequently accessed blocks have been introduced and implemented. These frequently accessed blocks, also called "high heat" blocks are placed in "hot" zones present in the center of the disk. In this chapter, based on earlier algorithms, we introduce a new technique to improve the disk response time. The basic idea in our work is to introduce multiple reorganized zones on disk instead of having a single one at the center of the disk. Using simulations of disk level traces, we found that these zones could improve the system response time by up to 30–50% for file server workloads when compared to the original block placement. With the disk performance increasing only at about 10% per year, this multiple zone strategy could match several years of progress in access time improvements.

## 3.2 Clustering Techniques

In this section we discuss the various existing block reorganization techniques studied earlier for deciding which blocks to be relocated. It has been observed that only a small fraction of data on a disk are used frequently. Out of millions of blocks present on a disk, just a few thousands of blocks absorb most of the requests. If these

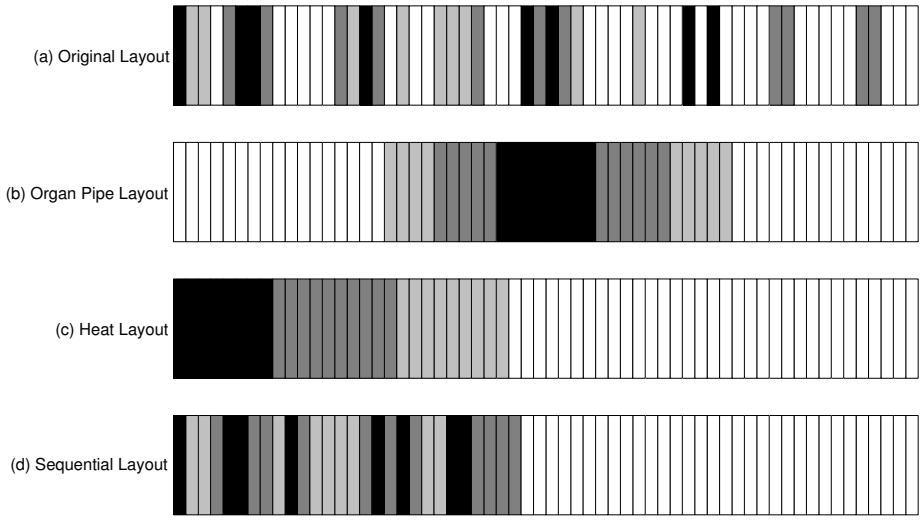


Figure 3.1: Disk Data Block Layouts

frequently accessed blocks are spread over the surface of the disk, far apart from each other, they would result in delayed seek time, causing poor disk performance (Figure 3.1(a)). These hot blocks could be clustered in a "hot" zone to reduce their seek times. This would enable the disks to take advantage of sequentially reading data subsequent to a request into a pre-fetch cache. This would further reduce the seek time, in case the order of data access remains in the order of their placement on disk surface. Many techniques to do this relocation of blocks have been suggested, a few of which we shall discuss in this section.

### 3.2.1 Organ Pipe

The most important result of work in this arena has been the organ pipe layout algorithm [Won, GS73, Knu98]. In the organ pipe algorithm, the most frequently accessed data, also referred to as "hot" blocks are placed at the center of the re-organized data, very similar to large organ pipes (Figure 3.1(b)). This is followed by lesser frequently accessed blocks being placed at the inner and outer edges of the center zone. This would minimize the head movement of the disk and works extremely well under smaller granularities. However, this technique works under the assumption

of considering the disk in one-dimensional space. Since disks are two-dimensional, this shuffling could also have negative impact on few workloads. Considering an example where a backward block is to be fetched - it would result in a complete rotation of the disk. This could also result in splitting contiguous data in case of workloads which could generate dependent references, which is done randomly [Smi85]. Thus, organ pipe layouts are usually optimal in practice, though, it could cause performance to fall in the storage system.

### 3.2.2 Heat Layout

To keep the head from scanning disks back and forth over the hot zone, it has been suggested to use the idea of heat layout [RW91], where the blocks are ordered in descending order of their heat or frequency of access (Figure 3.1(c)). It was seen that this method performed little better than organ pipe, but still failed when reorganizing small data blocks.

### 3.2.3 Packed Extents

In the earlier layout techniques, the original block sequence was not given consideration. Thus, the Packed Extents strategy [Hsu03], attempts to preserve the original block sequence, particularly under aggressive read-ahead. Their way of analyzing was to identify the most frequently accessed sequential units and then place them in ascending order of their ranking. Thus, the hot contents are packed together while maintaining their access sequence.

### 3.2.4 Sequential Layout

It was later observed that, removing less frequently accessed blocks from the packed extent model gave still better performance (Figure 3.1(d)) [Hsu03]. It was equivalent to placing the most frequently accessed blocks maintaining their order by

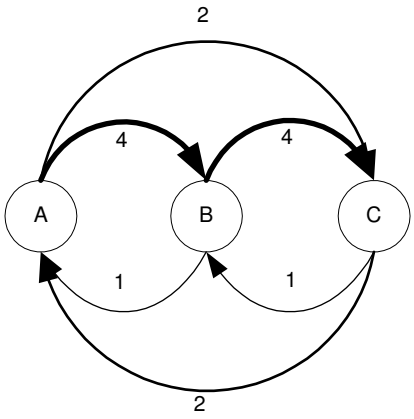


Figure 3.2: Allocation of Edge Weight in ALIS

their block numbers.

Other ideas such as placing the larger files on the outer zones were considered. If these large files are requested in their entirety, the increased transfer rate may reduce the overall latency. Also, due to large capacity of outer tracks, it will reduce the delays caused due to positioning of disk arm to read the entire file.

### 3.2.5 Automatic Locality – Improving Storage (ALIS)

The ALIS method, proposed by researchers at University of California, Berkeley and IBM in 2002, is an introspective storage system that automatically reorganizes selected disk blocks based on the dynamic reference stream to increase the spatial locality of reference and leverage the rapidly growing disk transfer rate [Hsu03]. It focuses on reducing the number of I/Os made on the disk by making the sequential prefetch more effective. Their algorithm considered having three copies of hot data, one being in the run clustering zone – given the highest priority while choosing which block to access, second in the heat-clustered zone, and if the data is not found in either, the originally existing data was used.

The first step in reorganizing blocks consisted of assigning weight to each edge generated while moving from one block on disk to another. The procedure of assigning

weight to these edges consisted of using graduated edge weight scheme (Figure 3.2). In this scheme, the weight of edge $i \rightarrow j$ is a decreasing function of the number of references between when those two data units are referenced [Hsu03]. It can be easily understood from the example shown in the figure. Let us assume the access sequence to be A,B,C,A,B,C. The weights are assigned based on the order in which the blocks were accessed. According to the algorithm, suppose $X_i$ denotes the reorganization unit referenced by the $i$-th read; for each $X_n$, we add an edge of weight $\tau$ - $j$ + 1 from $X_{n-j}$ to $X_n$, where $j \leq \tau$ [Hsu03]. In this sequence, move from A→B, B→C occurs twice. Hence, weight assigned to this edge would be two times of the weight for each consecutive jump. In our example, we set this value to be 2, and hence the weight of the edge from A→B and B→C is equal to 4. Weight of the following edges would be assigned one. It could thus be identified easily that A should be placed immediately followed by B and then by C for the referenced sequence.

For simple understanding, edge weight can be inferred as the number of times a reorganization unit is accessed within $\tau$ references of another. This value also represents the number of intermediate references between when the two units are accessed.

Once the weights for these edges are assigned, the access patterns should be interpreted. The following algorithm is adopted for finding the relocation run sequence $R$ [Hsu03]:

(1) Find the heaviest edge linking two unmarked vertices.

(2) Initialize $R$ to the heaviest edge found and mark the two vertices.

(3) Repeat

(a) Find an unmarked vertex $u$ such that

*headweight* = $\sum_{i=1}^{Min(\tau,|R|)}$ *Weight(u,R[i])* is maximized.

(b) Find an unmarked vertex $v$ such that

$tailweight = \sum_{i=1}^{Min(\tau,|R|)} Weight(R[|\ R\ |\ -\ i+1],v)$ is maximized.

(c) If $headweight > tailweight$

Mark $u$ and add it to the front of $R$.

else

Mark $v$ and add it to the back of $R$.

(4) End Repeat

## 3.3 Block Reorganization with Zone Layout

All the above strategies relocate the frequently accessed data blocks to the center of the disk. Relocating the blocks from any point on the disk surface to the center of the disk has an underlying assumption that the frequently accessed blocks are evenly distributed across the disk, and during the run, the jump from the normal block to the hot zone, average out to the center of the disk. Thus, placing the hot zone at the center of the disk is typically better than putting it at the inner and outer edges of the disk.

A single hot zone at the center of the disk assumes that nearly all accesses to the disk will be satisfied by the "hot" zone. In other words, once the disk head seeks to the hot zone it will never or rarely have to move off. Thus, seeks are minimized, if not completely eliminated. But, this might not be the case under all workloads. As we examined real workloads from the previous chapter and from [UMT], it became apparent that not all requests to the disk were satisfied by the hot zone. Typically, after blocks were relocated to the hot zone, still 20-50% of the subsequent block requests were not in the hot zone. For example, using the email server workload, if we moved all the blocks accessed over a 1 hour period to the hot zone, over the next hour, only 50% of those blocks were accessed. What this implies is that the

head must still make significant number of seeks to access requested data. Moreover, as it moves on and off the hot zone, any spatial locality that was intended by the file system is completely lost. In other words, if the file system had placed all the blocks of a particular file close together, and a block clustering scheme such as ALIS or sequential layout moves some portion of those blocks to the hot zone, as we make sequential requests to that file, we will actually incur extra overhead because the disk must go from the original location to the hot zone and back. This can be somewhat alleviated by keeping two copies of data, i.e. one in the original location, and another in the hot zone and have the disk always access the closest copy. However, this incurs extra space overhead and, in addition, our simulations show that in such a case, the disk rarely touches the hot zone to access the data since the closest copy is always at the original location.

In light of this phenomenon, it is clear that a single zone at the center of the disk is not ideal. Instead, we are proposing the use of multiple zones spread over the disk at regular intervals. We place hot zones every $x$ number of blocks, where $x$ is the *zone separation distance* and could vary from 30,000 to 10 million. For example, on a disk with 100 million blocks and a zone separation distance of 10 million, there would be 10 hot zones. When we relocate disk blocks, instead of relocating them to a single hot zone, at the center of the disk, we relocate them to the closest hot zone to the original location. By introducing multiple hot zones, the distance that the head must travel to get from the original location of data to the hot zone is reduced.

Using multiple zones takes advantage of the fact that locality of reference is both spatial and temporal. A single hot zone satisfies temporal locality in that it assumes a block accessed is likely to be accessed soon after and thus keeping the head on the hot zone. Using run-based mechanisms like ALIS can optimize for spatial locality but only if that locality of reference was seen before. By using multiple zones, we can have the temporal locality that hot zones provide while still maintaining spatial

locality because the hot zone is near the original location of the data. It does not depend on any a priori knowledge of the spatial locality of reference.

Splitting a single hot zone into multiple zones may raise concerns if frequently accessed blocks are spread across the disk and we find ourselves instead of moving from original location to a single hot zone, we are, instead, moving hot zone to hot zone. However, our simulations have shown that this is not an issue since the access patterns tend to concentrate on a particular region of disk before moving off to another region. Thus, the disk will make several small seeks off and on a particular hot zone before moving off to another hot zone. In a single zone system, these seeks on and off the zone would be much larger.

We evaluated the efficacy of the multiple zone method in concert with two clustering techniques – ALIS and sequential layout. We called the ALIS based method ALIZ (ALIS with zones) and the sequential layout based method FREQZ.

## 3.4 Architecture of Zone Layout Schemes

It is conceivable that these zone layouts could be implemented at various levels within the disk architecture of the system. It could be added to the disk driver, and could also be introduced at the file system level. However, it must be noted that ALIZ, just like ALIS, requires heavy processing power. Hence, its presence within the system could lay an extra burden on the system. To avoid this, ALIZ could be implemented as a separate component with its own processing power. On the contrary, FREQZ algorithm is faster, and does not require much processing. Hence, its implementation within the driver, or at the file system level would be easier.

## 3.5 Results

### 3.5.1 Simulation Methodology

We evaluated the zoned block reorganization method using simulations. Our simulation model consists of two systems – a single disk system and a RAID system. We took a disk level trace in the **ascii** format [GGP99] and modified it according to the specifications of the ALIZ or FREQZ algorithm. The result of the modification is a new trace file that reflects the reorganization of the blocks, which should have been accessed if the algorithm was already implemented into the system. In our simulation, we tested the algorithm with reorganization taking place every 15 minutes. This however could be set to a higher value also. On obtaining the simulated trace, it was given as input to DiskSim [GGP99], a highly configurable disk simulator.

In our study, we used a part of trace of email server from the previous chapter and also a database server. The database workload was obtained from the UMass trace repository [UMT]. This trace was obtained from the disk accesses of a large financial institution consisting of online transaction processing applications. It acts as a common database server, and hence consists of good amount of read and write access made to the disk. The characteristics of both traces are shown in Table 3.1.

| | Email Server Workload | Database Workload |
|---|---|---|
| Number of reads (%) | 92 (0.2%) | 4099354 (76.84%) |
| Number of writes (%) | 45947 (99.8%) | 1235633 (23.16%) |
| Length of trace (s) | 12350 | 5000 |

Table 3.1: Trace Characteristics

We conducted experiments on different traces and with four different algorithms and two different disk systems. We tested results with the performance under the original trace, traces with ALIS relocation, ALIZ relocations, FREQNC and finally FREQZ relocation algorithms. In this thesis, we refer to the sequential layout scheme

as FREQNC. For the zone based algorithms, we also evaluated various zone separation distances, varying from 30,000 blocks to 10,000,000 blocks. For the most part, zone separation distances of 30,000 blocks were too small and caused significant degradation in performance and, therefore, are not shown on any of the graphs. Note, also, that these blocks are disk blocks or sectors, or 512 bytes per block.

Under the ALIZ algorithm, we noticed approximately 1500-1700 blocks being moved at the end of each iteration for the email server workload. However, for the database server workload there were 45,000 blocks on average being relocated at the end of each tracing period. For the FREQZ algorithm, as expected, there were higher number of blocks being relocated. For the email server we noticed approximately 25,000 blocks being relocated every time the relocation algorithm was applied, while for the database server workload FREQZ moved almost 50,000 blocks.

### 3.5.2 Single Disk Results

The single disk system that we simulated was a drive with 16,383 cylinders, 3120 sectors per cylinder, and 14 heads. The average seek time is 7.35 ms and the rotational latency is 8.3 ms due to the spin speed of 7200 RPM. The total capacity of the disk is 41,174,138,880 bytes.

#### 3.5.2.1 ALIZ on Single Disk

We tested our algorithm with the two traces discussed earlier using the above mentioned methodology. With two different kind of workloads, we try to establish how much improvement in performance using ALIZ could be gained over the original layout.

**Email Server (Read Dominated Workload)**

From Table 3.2 and Figure 3.3, we can see that ALIS and ALIZ perform equally well on the email server workload, with ALIZ having much better results for multiple

zone separation distances of 50K and 100K. The algorithms were seen to show an improvement by almost a factor of one-third with respect to the original system block organization. This however failed to maintain the same improvement flow for larger zone separation distances. The smaller zone separation distance are ideal for the email server workload, since the locality regions are much smaller. In other words, with the email server, accesses that are clustered together tend to access regions that are smaller. This is most likely because emails are typically just a few blocks in size.

| | **System Response Time** | **% Improvement** |
|---|---|---|
| **Original** | 241.182100 | — |
| **ALIS** | 173.821372 | 27.93 |
| **ALIZ (50k)** | 152.378009 | 36.82 |
| **ALIZ (100k)** | 166.424039 | 31.00 |
| **ALIZ (500k)** | 179.432142 | 25.60 |
| **ALIZ (1m)** | 239.829288 | 0.56 |
| **ALIZ (10m)** | 224.313921 | 6.99 |

Table 3.2: System Response Time for different clustering technique with ALIZ on single disk for Email Server Workload

**Database Server (Read-Write Workload)**

Database server workload results are shown in Table 3.3 and Figure 3.4. On the database workload, ALIZ performed much better than ALIS. Though there is a performance loss for ALIZ with a 1m zone separation distance, we, however, see a gain of about 35% for smaller zone separation distances. This is where we see a real advantage of having multiple "hot zones" . As with the email server workload, the reason is that most of the requests in database workloads are small in size and the accesses on them are clustered. Thus, jumps from the original block location to the high heat block would be must shorter when there exists multiple zones.

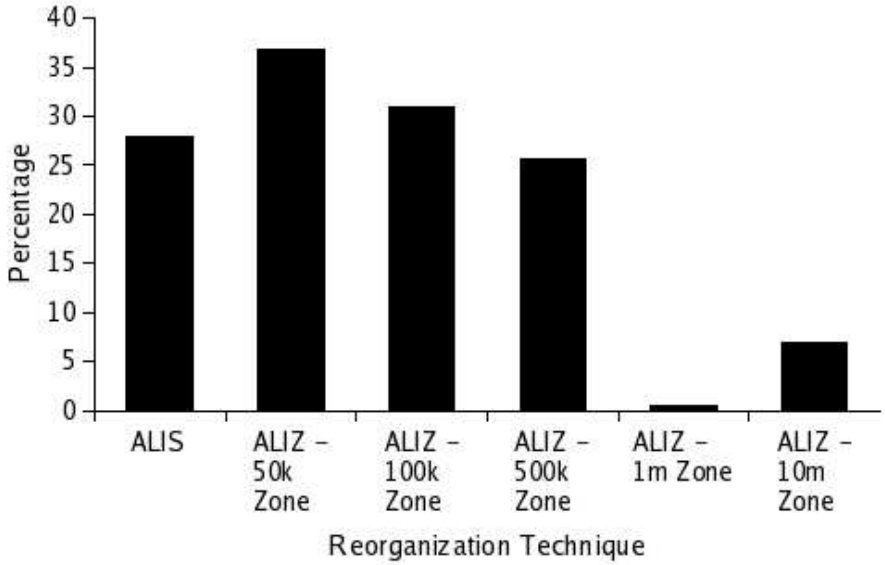


Figure 3.3: Percentage improvement in System Response time for different clustering techniques with ALIZ on single disk under Email Server Workload

| | **System Response Time** | **% Improvement** |
|---|---|---|
| **Original** | 56.525077 | — |
| **ALIS** | 53.573672 | 5.22 |
| **ALIZ (30k)** | 41.791646 | 26.07 |
| **ALIZ (50k)** | 36.468853 | 35.48 |
| **ALIZ (100k)** | 43.719855 | 22.65 |
| **ALIZ (1m)** | 60.463968 | -6.97 |
| **ALIZ (10m)** | 53.392433 | 5.54 |

Table 3.3: System Response time for different clustering technique with ALIZ on single disk for Database Workload

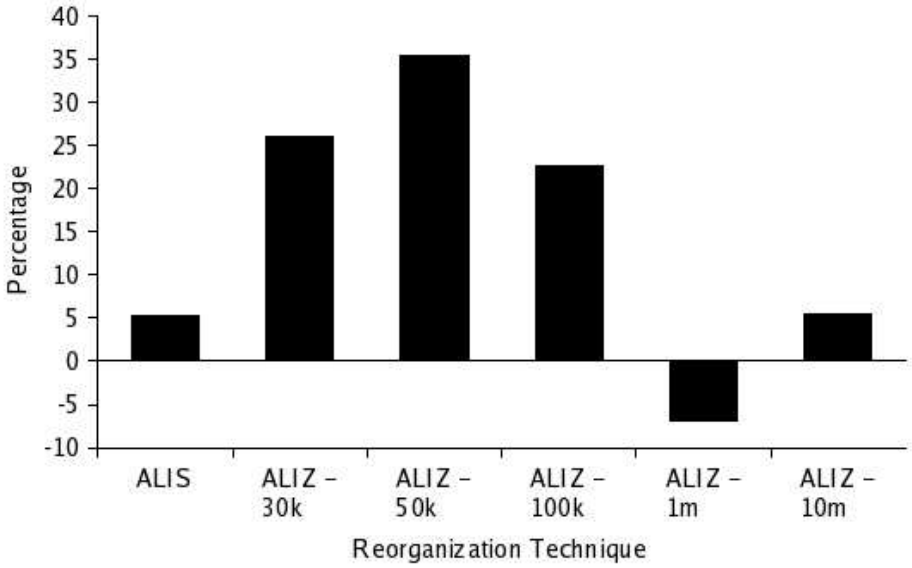


Figure 3.4: Percentage improvement in System Response time for different clustering techniques with ALIZ on single disk under Database Workload

#### 3.5.2.2 FREQZ on Single Disk

We test the same workload using the FREQZ algorithm, which would place all the accessed blocks into a hot zone.

**Email Server (Read Dominated Workload)**

For the email server workload, FREQZ showed no considerable improvement in performance. It showed results very close to that under the original layout of blocks. This is because there is very little temporal locality of reference in a email server. A mail spool file is user specific and is typically only accessed once or twice a day, thus it is unlikely that particular data block will be seen with any frequency. Thus, relocating every data block would generate the same pattern as was already existing in the normal layout. The results for FREQZ on the Email server is shown in Table 3.4 and Figure 3.5.

| | **System Response Time** | **% Improvement** |
|---|---|---|
| **Original** | 241.182100 | — |
| **FREQNC** | 240.625690 | 0.23 |
| **FREQZ (50k)** | 241.858953 | -0.28 |
| **FREQZ (100k)** | 240.796281 | 0.16 |
| **FREQZ (500k)** | 240.371581 | 0.34 |
| **FREQZ (1m)** | 240.659394 | 0.22 |

Table 3.4: System Response time for different clustering technique with FREQZ on single disk for Email Server Workload

**Database Server (Read-Write Workload)**

Table 3.5 and Figure 3.6 show results of FREQZ for the database workload. The FREQZ algorithm is significantly better than the base sequential layout. At a zone separation distance of 50,000 blocks, FREQZ is nearly 40% better than sequential layout. In fact, FREQZ is nearly as good as ALIZ, and also better than ALIS. This gain is because the blocks are accessed in clusters and hence, placing them closer reduces the disk access time.

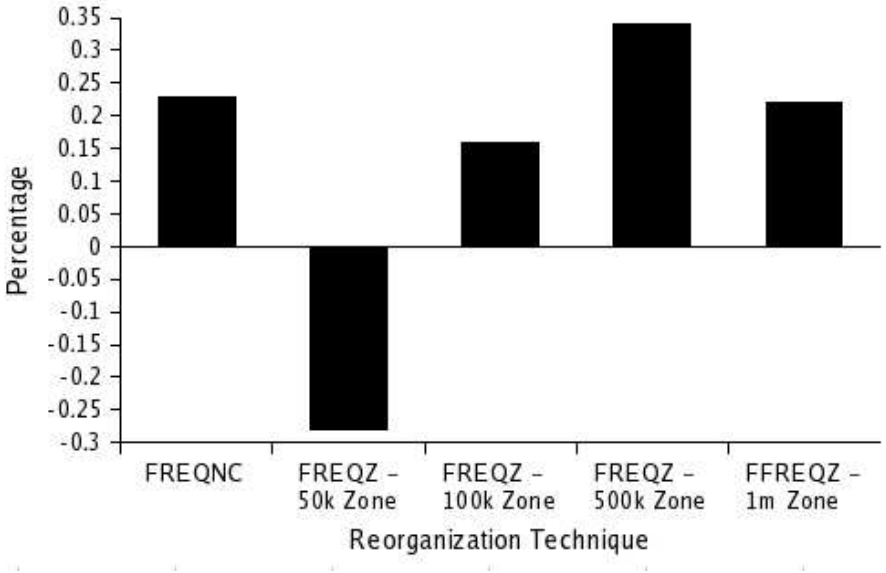


Figure 3.5: Percentage improvement for different clustering techniques with FREQZ on single disk under Email Server Workload

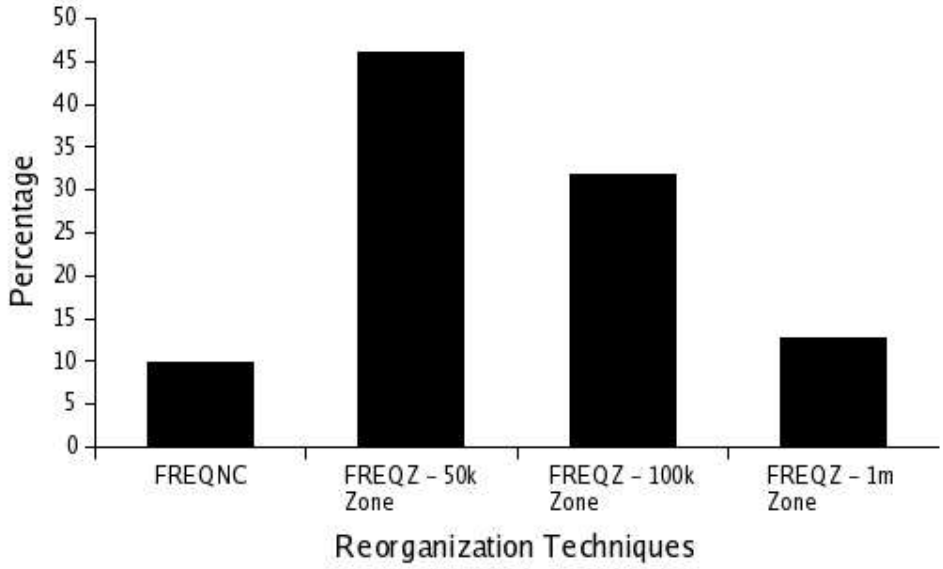


Figure 3.6: Percentage improvement for different clustering techniques with FREQZ on single disk under Database Workload

| | System Response Time | % Improvement |
|---|---|---|
| **Original** | 56.525077 | — |
| **FREQNC** | 50.895483 | 9.96 |
| **FREQZ (50k)** | 30.463855 | 46.11 |
| **FREQZ (100k)** | 38.596019 | 31.72 |
| **FREQZ (1m)** | 49.375421 | 12.65 |

Table 3.5: System Response time for different clustering technique with FREQZ on single disk for Database Workload

### 3.5.3 RAID System Results

We used the same algorithm as explained in the previous section for inspecting their performance on RAID systems. Conducting various tests on different workloads, with different algorithm, we find out that these algorithms perform really well on RAID systems also. The RAID systems that we simulated consisted of 9 disks arranged in a RAID5 organization. We also examined the effect of stripe size on the results and varied that from 2 sectors to 512 sectors. Again, we compare the results of the original trace with the ALIS, ALIZ, FREQNC and FREQZ clustering algorithms.

We assume that RAID is treated as a monolithic single disk. Thus, the hot zone is spread across all nine disks of the array. For a single zone strategy, the hot zone area is still located at the center of each disk and hot blocks become striped across the array. For a multiple zone system as we are proposing, the net effect is that the zone separation distance is reduced by a factor equal to the number of disks in the array. Thus, a 1 million zone separation distance on a nine disk RAID is equivalent to zones being 111K blocks apart on each disk.

In this section, we take a look at the results which we obtained from the implementation of zones on RAID, on already existing reorganization techniques. We here compare the results of the original trace with ALIS, ALIZ, FREQNC and FREQZ. We also consider the output at the different raid stripe sizes.

#### 3.5.3.1 Email Server (Read Dominated Workload)

**System Response Time**

For the Email server workload on RAID, both ALIS and ALIZ offer significant performance gains, as can be seen from Figure 3.7 and Figure 3.8. However, benefit of ALIZ over ALIS is not as clear as it was with a single disk. On stripe sizes of 32 and 64, which are typical stripe sizes, ALIZ does have a slight advantage, but on other strip sizes the difference is less noticeable. The reason is that the hot zones striped across the array. For example, if there is a cluster of 5 spatially close accesses, in a single disk those accesses will be kept close together because the hot zone is nearby in a multiple zone system. However, in a RAID system, those 5 accesses advantage of keeping the head at the same place for an extended period of time. In other words, we only access one or very few blocks at a time on the RAID system. This is particularly apparent when we look at the sequential layout based algorithms on RAID.

In Figures 3.9 and Figures 3.12, we see that the higher the zone separation distance, the better the performance. This is especially true on ALIZ. The reason is that higher zone separation distance works better on RAID is because we are treating the RAID as a monolithic address space. In other words, a 50K zone separation distance at the RAID system level becomes 5555 blocks at the disk level. This zone separation distance is much too small and can decrease the performance significantly. Primarily, this is because with the zone distances so small, the clusters over which we are trying to find spatial locality is too small. For ALIZ, these small zone separation distances are troubling because they make it difficult to find decent runs. Moreover, introducing such frequent zones could have separated the large sequences, responsible for poor performance.

Figures 3.11 and 3.12 shows the poor performance variation of FREQZ layout algorithm with the original layout. On FREQZ also, we notice a very similar behavior as on ALIZ. The algorithm showed improvement on the lower and most common stripe

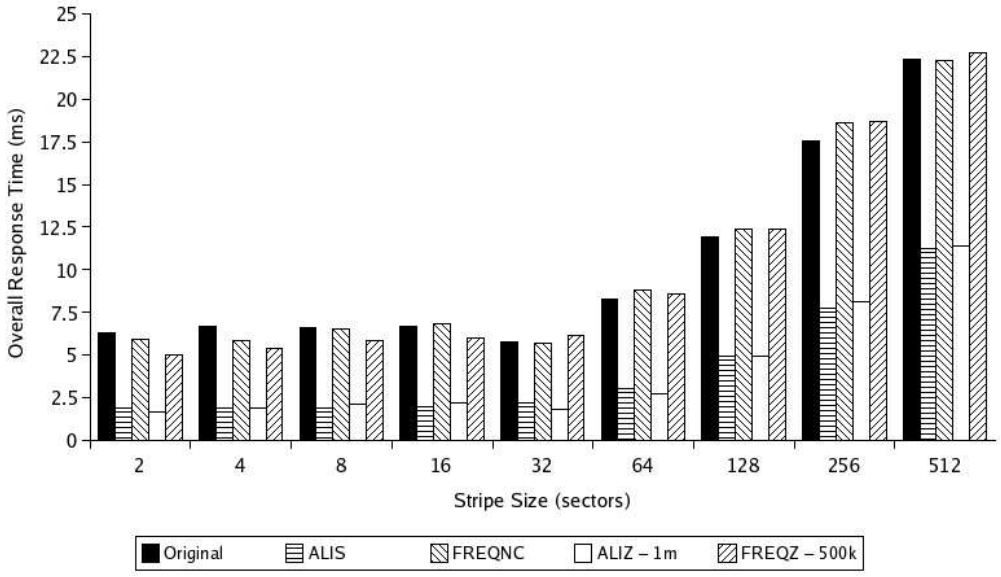


Figure 3.7: System Response time for Email Server Workload (RAID System)

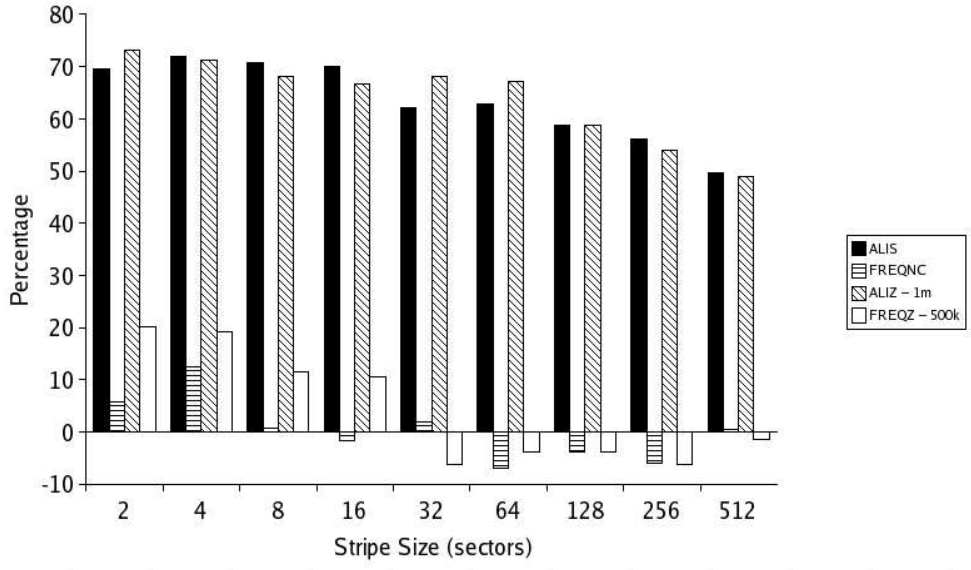


Figure 3.8: Percentage improvement in System Response time for Email Server Workload (RAID System)

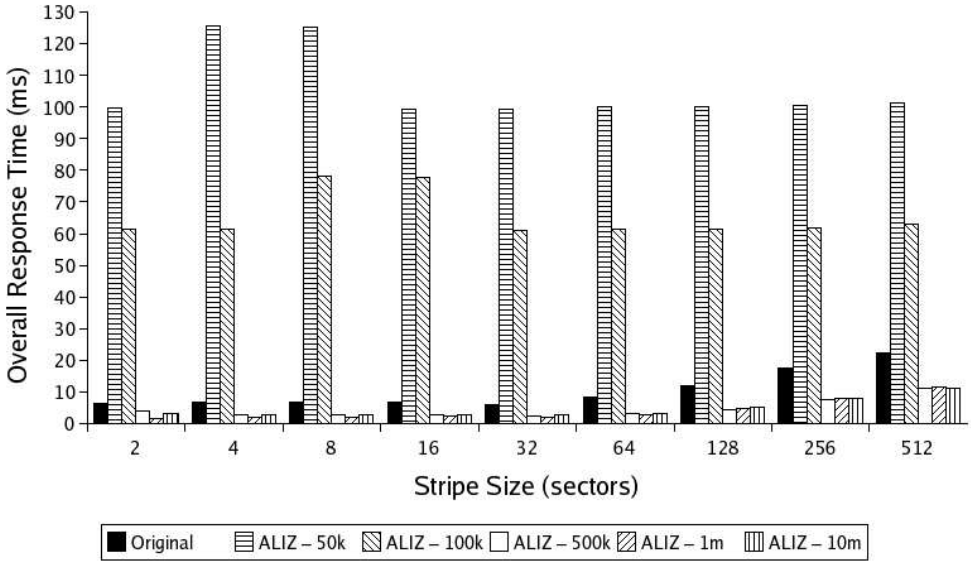


Figure 3.9: System Response time for ALIZ with different zone separation distances under Email server Workload (RAID System)

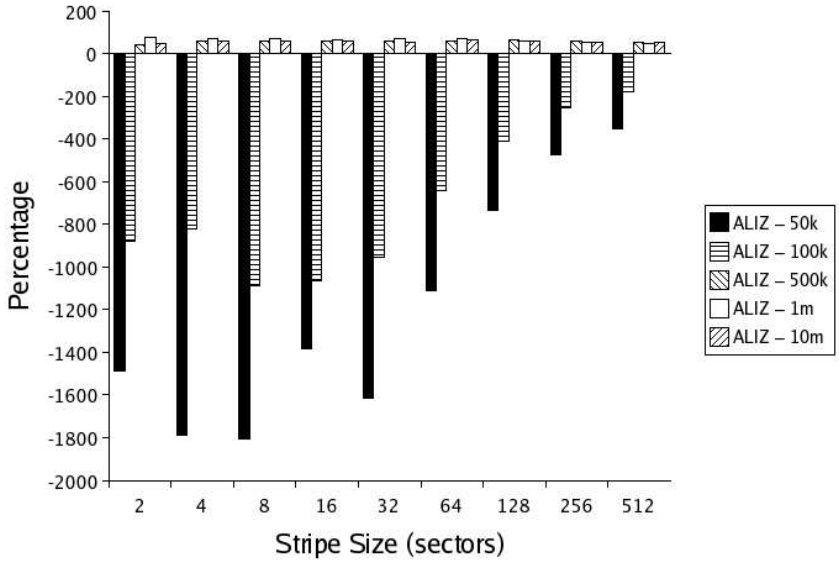


Figure 3.10: Percentage improvement in System Response time for ALIZ with different zone separation distance under Email Server Workload (RAID System)

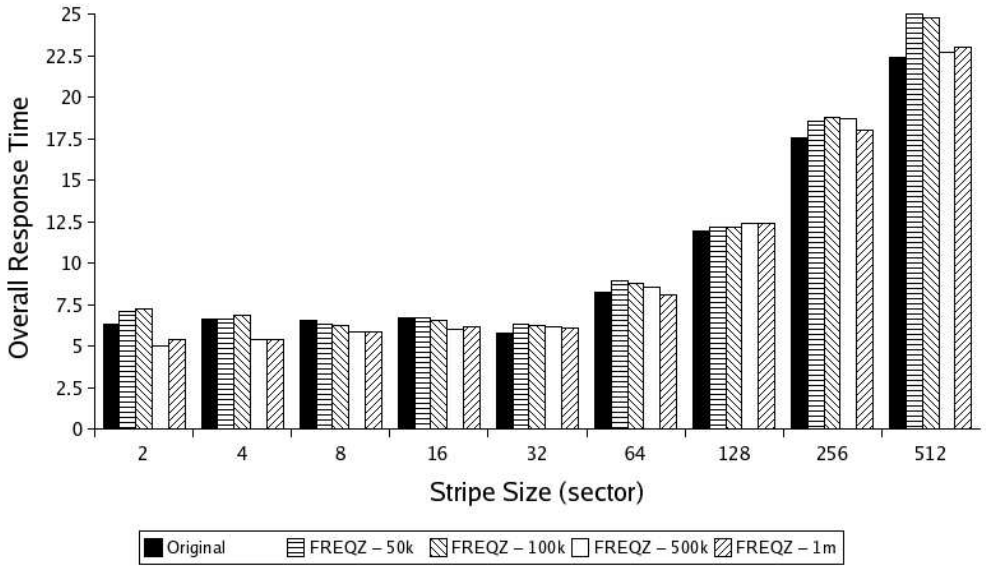


Figure 3.11: System Response time for FREQZ with different zone separation distances under Email Server Workload (RAID System)

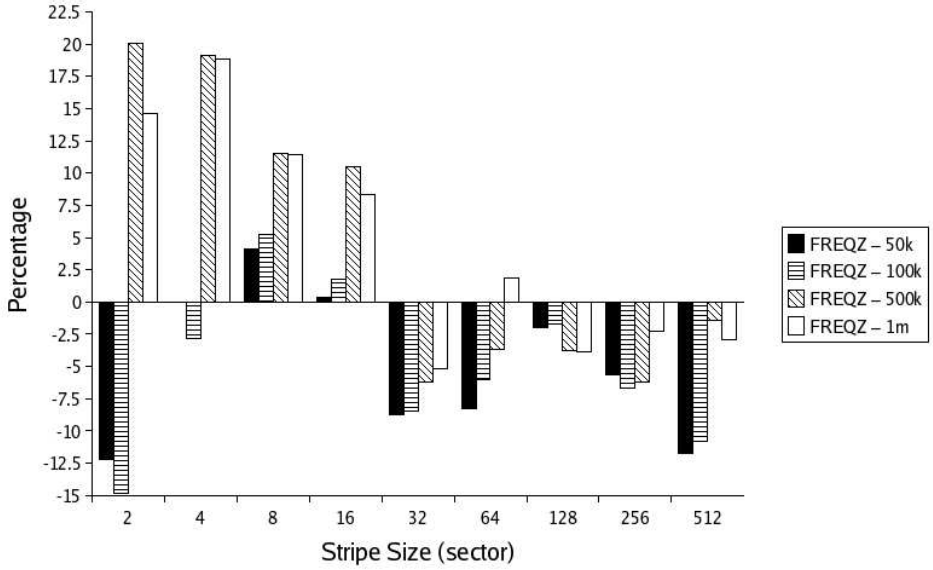


Figure 3.12: Percentage improvement for FREQZ with different zone separation distances under Email Server Workload (RAID System)

sizes. We see up to 20% improvement in performance when compared to the original layout for the lower stripe sizes. However, FREQZ could not give as good performance as ALIZ. This is because, the pruning method in FREQZ does not consider the order in which the data was accessed. However, FREQZ showed improvement against its single zone layout arrangement FREQNC.

**Average Seek Distance**

Figure 3.13 – 3.18 shows the Average Seek Distance on disk for the various reorganization schemas. This follows a very similar pattern as that of the system response time, with the performance being better for the lower stripe sized RAID systems. For higher stripe sized RAID systems, we see that the seek time increases by almost 5-10%. However, we notice a very high reduction in the average seek distance for the FREQZ layout. Even though we see a very high reduction in the disk seek time for FREQZ, such high performance improvement on the system response time is not noticed.

FREQZ showed large reduction in average seek distance in the commonly used 32 and 64 sector striping as can be seen in Figure 3.17.

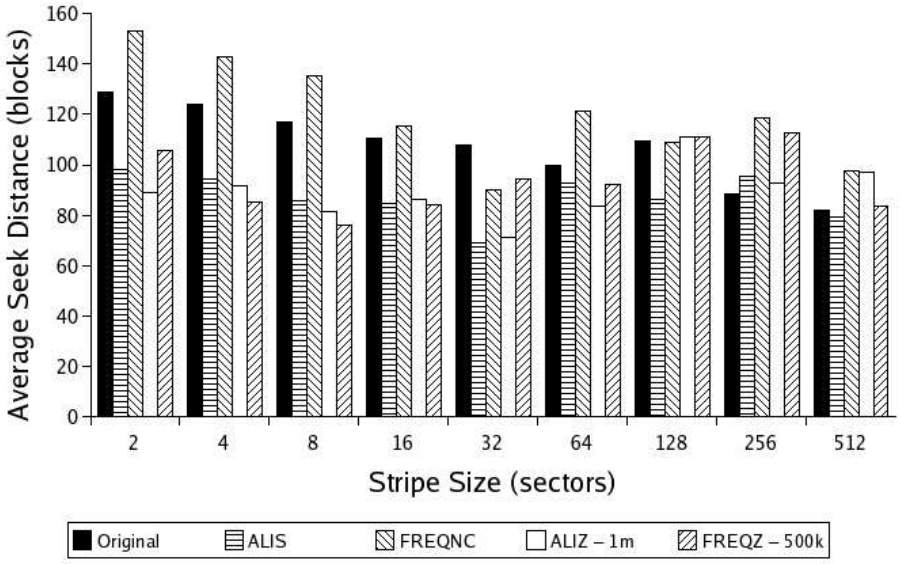


Figure 3.13: Average Seek Distance for Email Server Workload (RAID System)

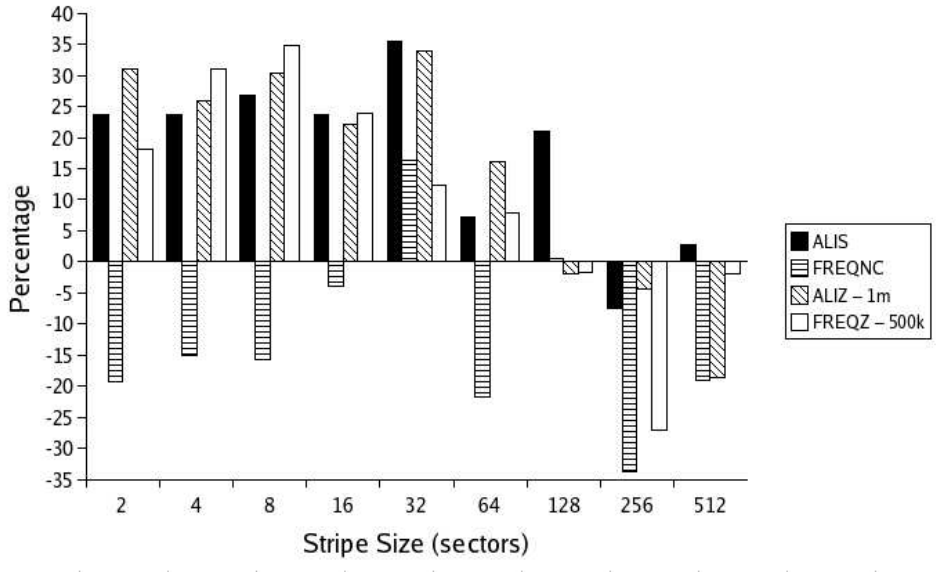


Figure 3.14: Percentage improvement in Average Seek Distance for Email Server Workload (RAID System)

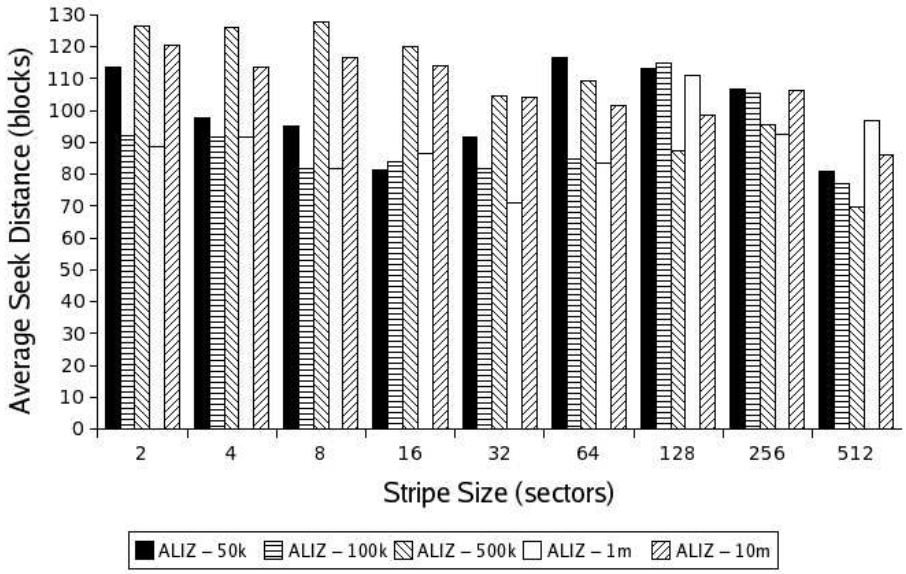


Figure 3.15: Average Seek Distance for different zone separation distance on ALIZ under Email Server Workload (RAID System)

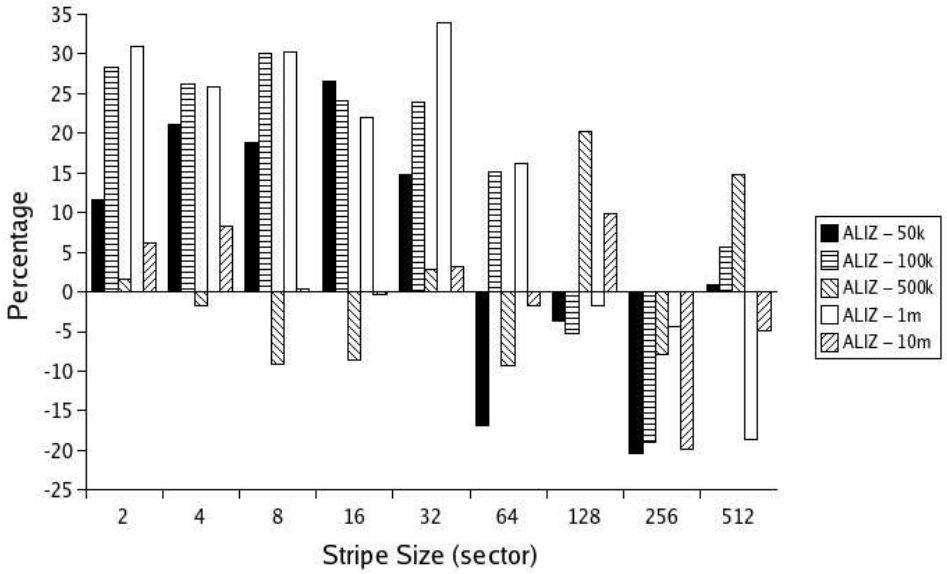


Figure 3.16: Percentage improvement in Average Seek Distance for ALIZ under Email Server Workload (RAID System)

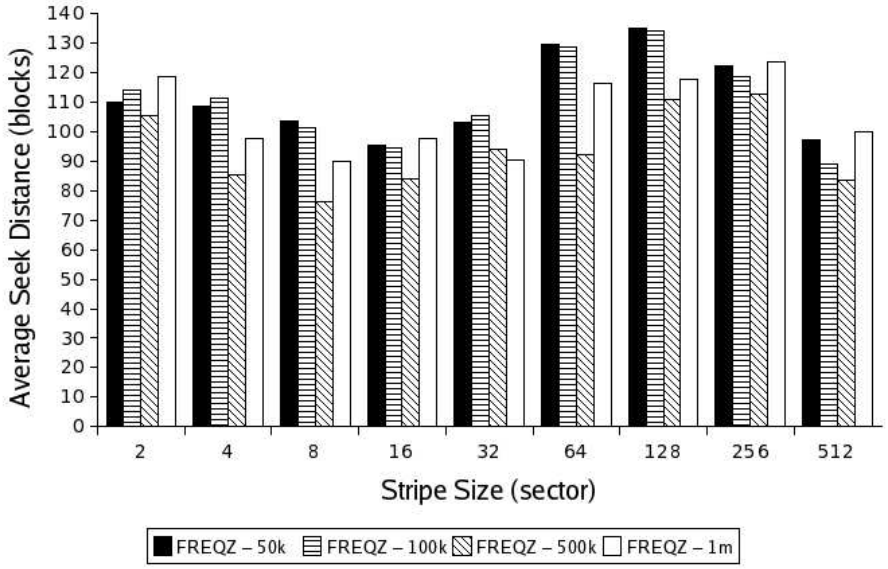


Figure 3.17: Average Seek Distance for different zone separation distance on FREQZ under Email Server Workload (RAID System)

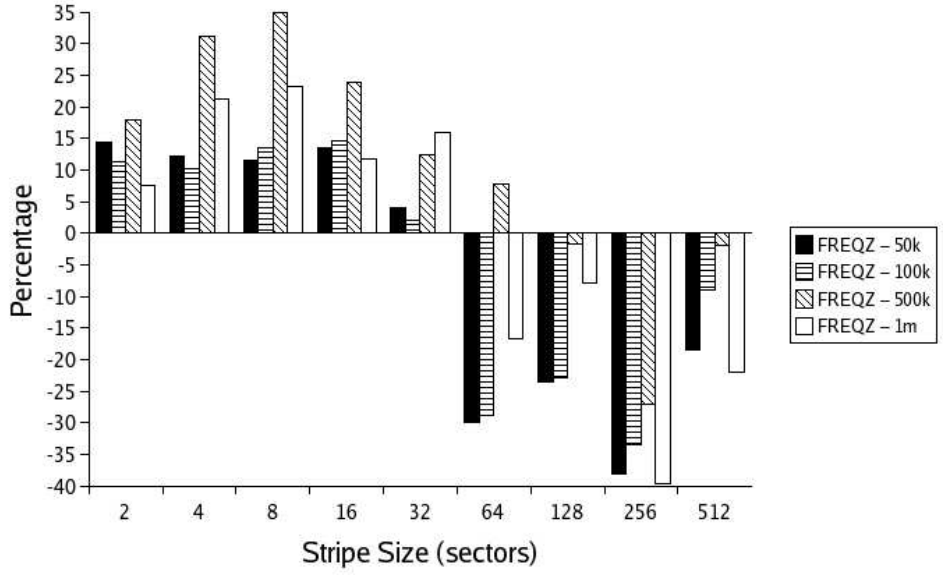


Figure 3.18: Percentage improvement in Average Seek Distance for different zone separation distance on ALIZ under Email Server Workload (RAID System)

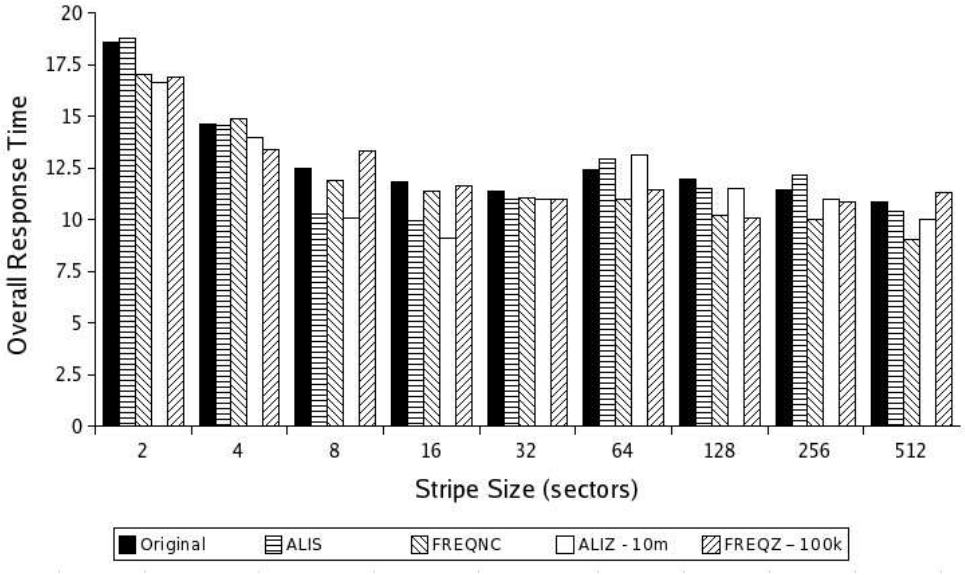


Figure 3.19: System Response time for Database Server Workload (RAID System)

#### 3.5.3.2 Database Server (Read-Write Workload)

**System Response Time**

The Financial database workload consists of approximately 75% read requests. Under this workload, ALIZ again performs better than ALIS on the lower stripe sizes. FREQZ also performs better than FREQNC for lower stripe sized RAID systems. Figure 10 shows the percentage improvement in the overall response time of the different layouts. Using ALIZ, we could see performance improvement of up to 20% for stripe size of 16, but it showed negative results for higher stripe sizes, with percentage improvement for stripe size of 512 going worse by a factor of 18% (Figure 3.21 and Figure 3.22).

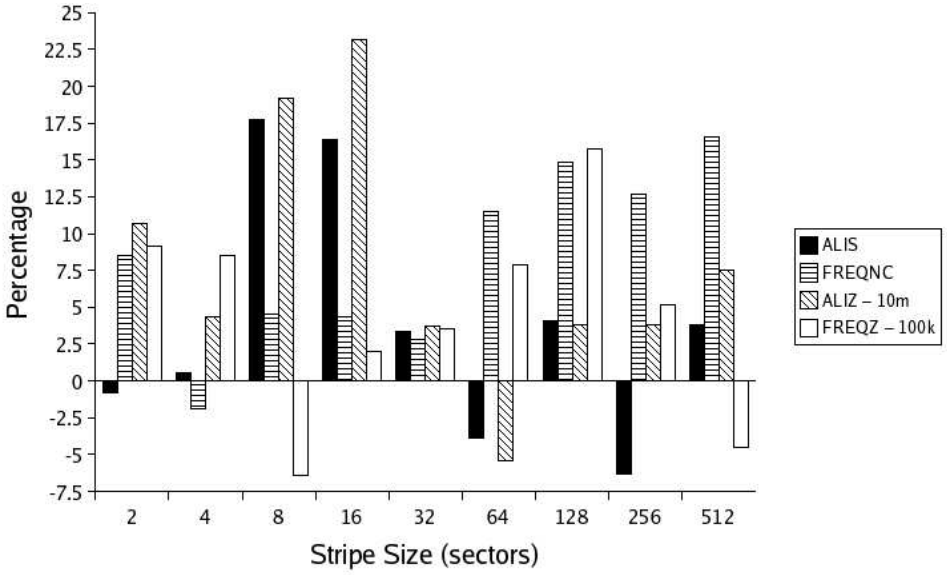


Figure 3.20: Percentage improvement in System Response time for Database Server Workload (RAID System)

Larger zone separation distance on FREQZ under the database workload showed better response time (Figure 3.23 and 3.24). This nature is because the database workload consists of smaller requests, and most of these requests are present as clusters. By applying the FREQZ algorithm, we place the frequently accessed blocks closer in the hot zone, and hence, make the disk access faster. This is the reason why

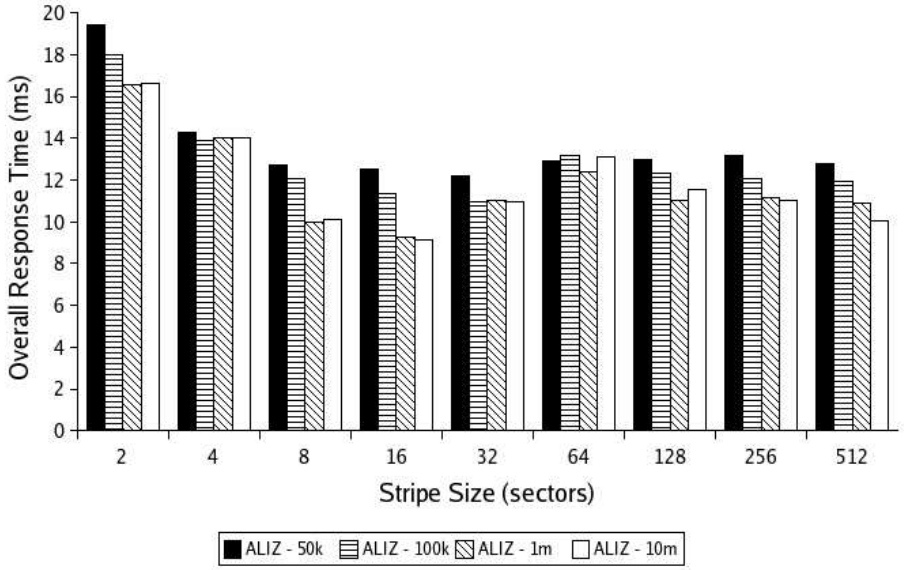


Figure 3.21: System Response time for different zone separation distances on ALIZ under Database Server Workload (RAID System)

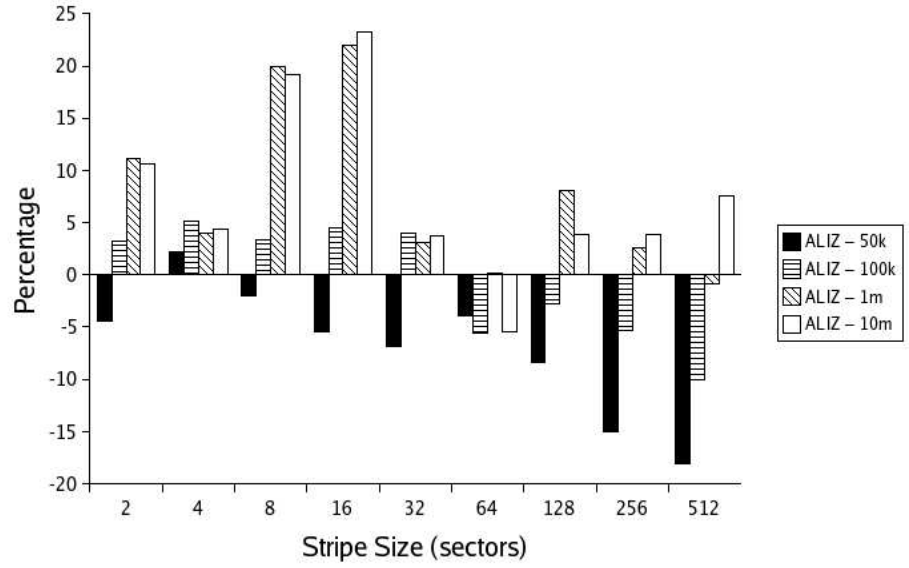


Figure 3.22: Percentage improvement in System Response time for different zone separation distances on ALIZ under Database Server Workload (RAID System)

we do not see a good improvement for smaller zone separation distances, since they are not sufficient to bring most of the sequential access together.

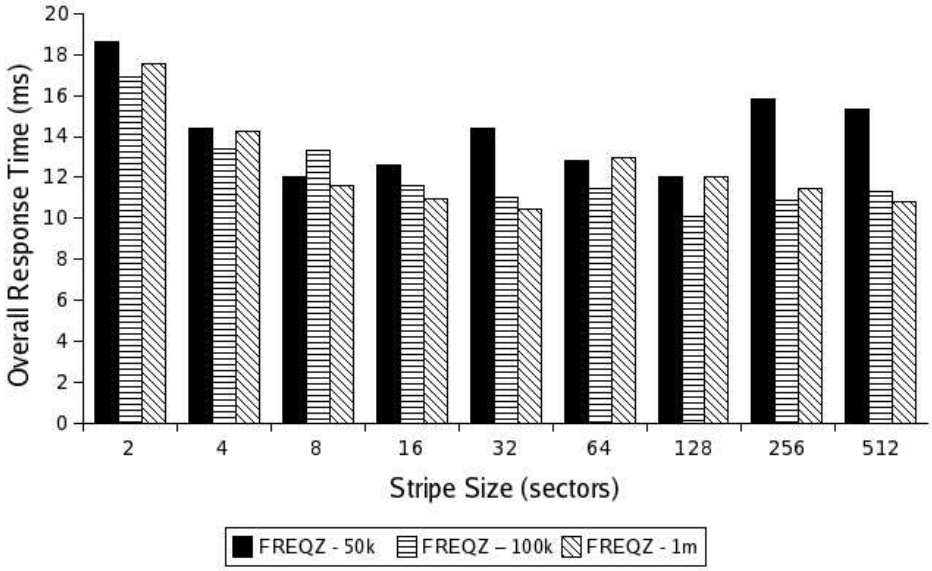


Figure 3.23: System Response time for different zone separation distances on FREQZ under Database Server Workload (RAID System)

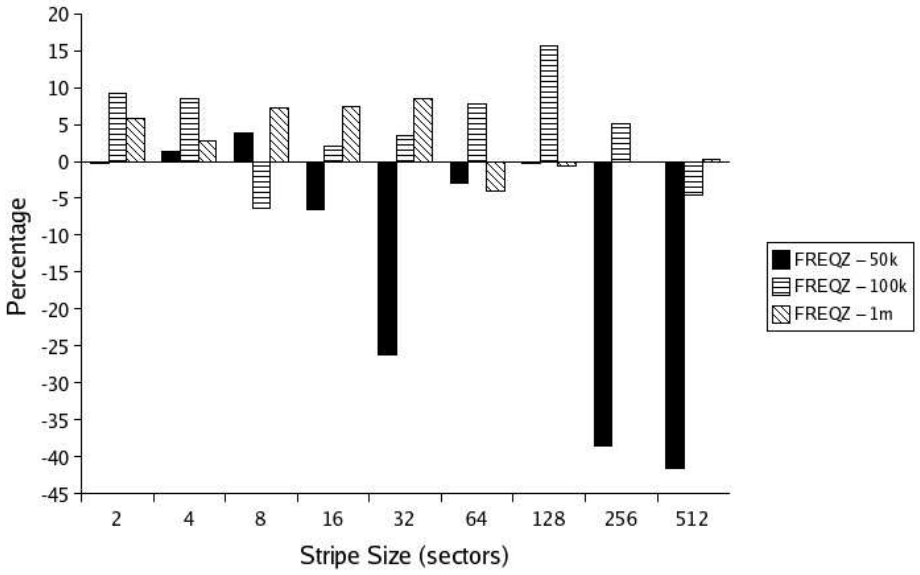


Figure 3.24: Percentage improvement in System Response time for different zone separation distance on FREQZ under Database Workload (RAID System)

### Average Seek Distance

Average disk seek distance did not show much improvement under the database work-

load. Figure 3.25 and Figure 3.26 shows almost equal seek distance for almost all of the schemes, except for stripe size of 8, where ALIZ and ALIS performed well, bringing about 20-25% improvement.

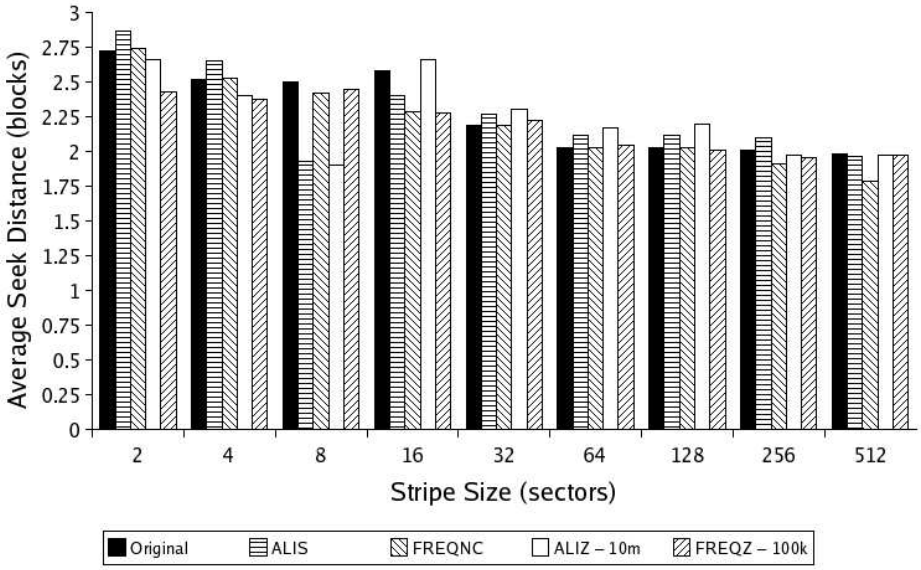


Figure 3.25: Average Seek Distance for Database Server Workload (RAID System)

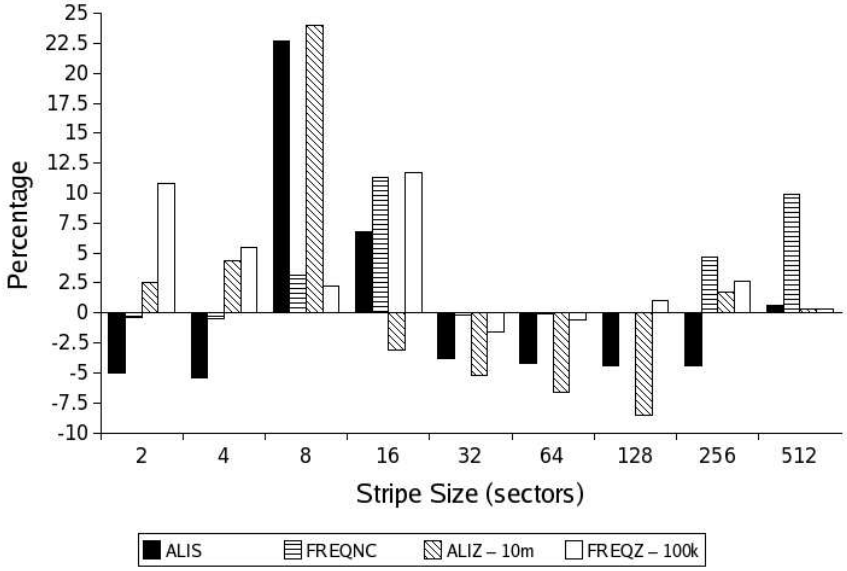


Figure 3.26: Percentage improvement in Average Seek Distance for Database Server Workload (RAID System)

ALIZ did not show very good improvement in the disk seek distance under the

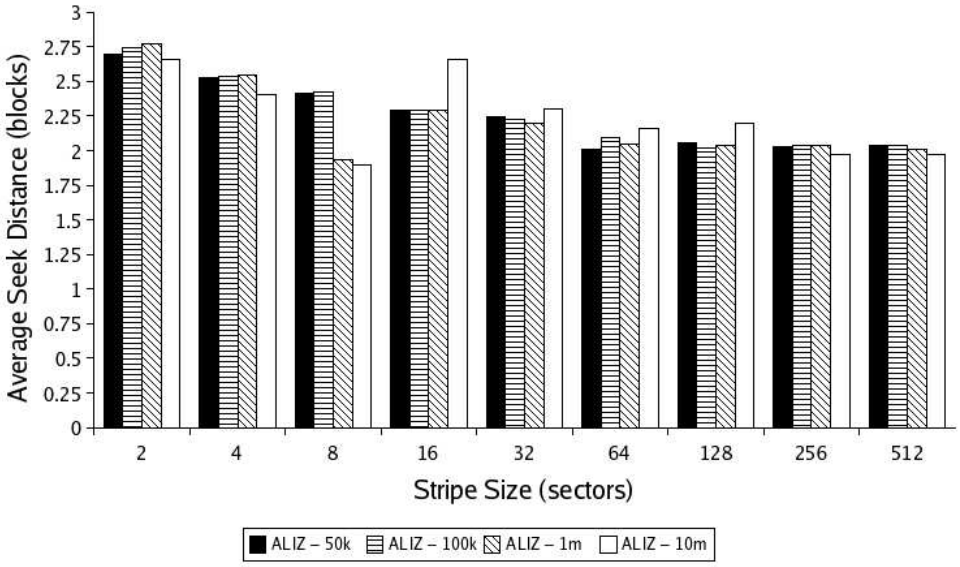


Figure 3.27: Average Seek Distance for different zone separation distances on ALIZ under Database Server Workload (RAID System)

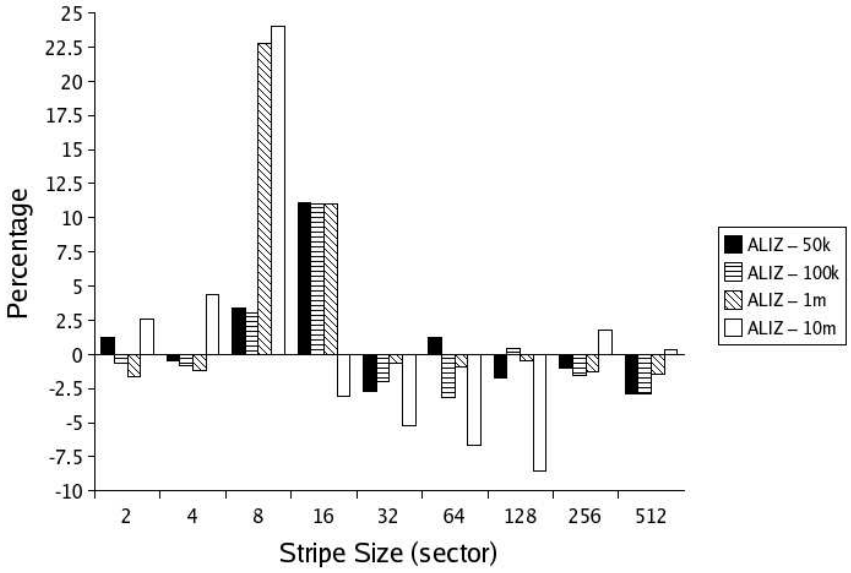


Figure 3.28: Percentage improvement in Average Seek Distance for different zone separation distances on ALIZ under Database Server Workload (RAID System)

database workload.

FREQZ however, consistently showed improvement in the disk seek distance, with zone size of 1 million blocks giving very good performance. Gain of almost 20% was achieved for stripe size of 16, while an average gain of 10-12% was maintained for

larger stripe size (Figure 3.29 and Figure 3.30).

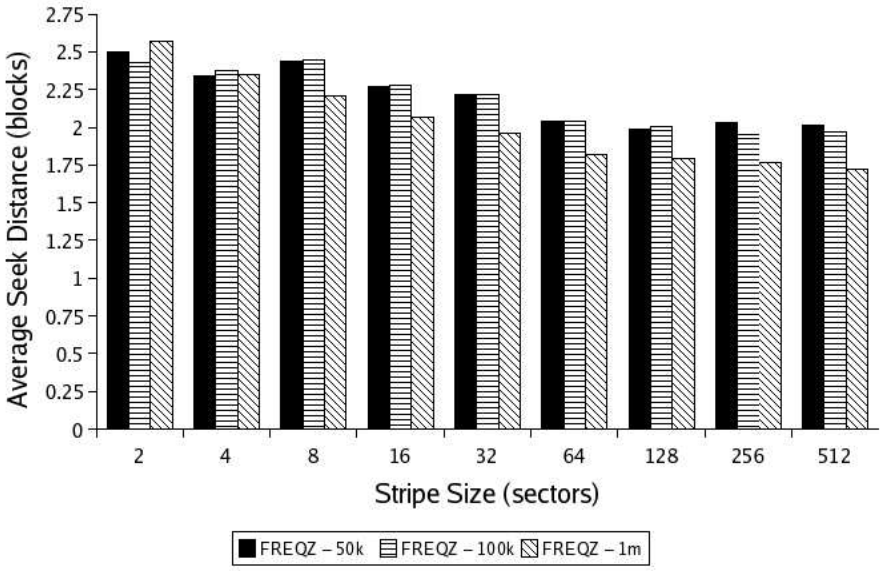


Figure 3.29: Average Seek Distance for different zone separation distances on FREQZ under Database Server Workload (RAID System)

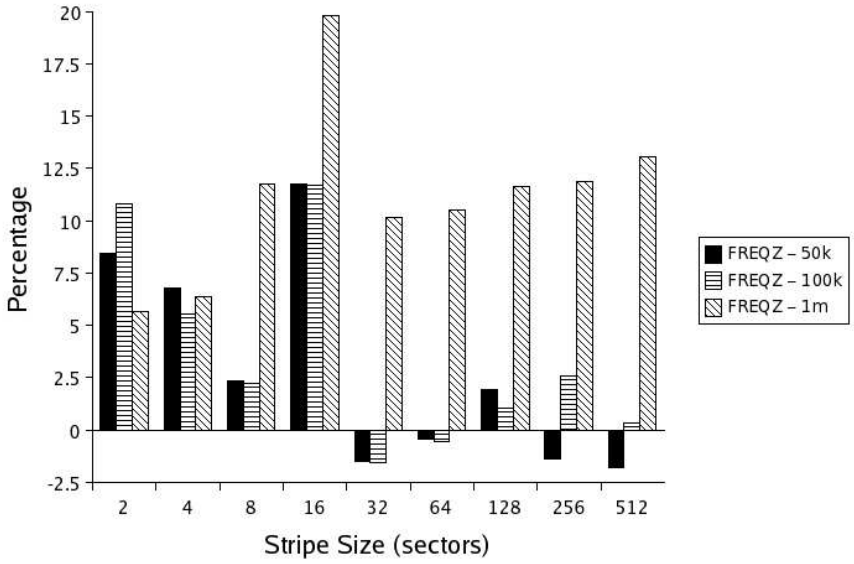


Figure 3.30: Percentage improvement in Average Seek Distance for different zone separation distances on FREQZ under Database Workload (RAID System)

## 3.6 Summary

In this chapter, we have presented a multiple zone block reorganization method to improve disk system response time. We see 10-40% improvement over existing block reorganization schemes in single disk settings. However, in RAID systems, the advantage of multiple zone is lost because the zone is spread across the array. To address this, it is better to employ multiple zones at the individual disk level rather than across the array. The performance of multiple zone clustering seems to be best at a 50,000 blocks zone separation distance for single block systems. However, it is clear that depending on the workload the optimum may change. This is particularly true for RAID systems. In light of this, we envision that the zone separation distance could be determined dynamically based on the workload or perhaps statically with a tool that analyzes profiled traces.

# Chapter 4

# Conclusions

In this thesis, we explored the various file systems performance and their optimization techniques. From Chapter 2, we gather that the file system has a very minor effect under read-dominated workloads. They do not have a major role to play in the disk performance. However, a noticeable difference was seen between the non-journaling and the journaling file systems, where journaling file system made lot more access to the disk. JFS and ReiserFS seem more efficient journaling file systems with both of them making lesser accesses to the disk (Table 2.2). Because of such large amount of difference in the number of accesses made on the disk due to journaling, use of NOVRAM is suggested for collecting these transaction logs. NOVRAM contains its memory even if on power loss. A close look at the file system performance showed JFS providing more idle time at the disk level in comparison with other journaling file systems – EXT3 and ReiserFS. This idle time could be used for the optimization techniques such as those discussed in the Chapter 3.

In Chapter 3, we discussed the new multiple zone layout strategy, which improved the system response time by 10-40% over the already existing reorganization techniques. We however observed that under certain cases, we had to pay the penalty of using these multiple zones. However, since these techniques are workload dependent, we anticipate a design of a utility, which could dynamically analyze the workload and

suggest an optimum zone separation distance and stripe size for the RAID systems.

## 4.1 Future Work

In our work, we observed a significant improvement in the system response time using the ALIZ algorithm over the FREQZ algorithm. However, its use is being limited by the processing power of the system. This is because of the large number of edges which are under consideration by the algorithm even after removing the lower 10 percentile edges from the edge list [Hsu03]. Due to this, its use requires a separate component with its own computation capability. We plan to mix the packed extent layout technique with the ALIS algorithm to reduce the edge count and hence increase its processing speed. Reduction in its processing power might help its easy implementation at the driver level for improving the system performance.

# Bibliography


[AS95] Sedat Akyurek and Kenneth Salem. Adaptive block rearrangement. In **ACM Transactions on Computer Systems**, number 13(2):89–121, 1995.

[BC03] Daniel P. Bovet and Marco Cesati. **Understanding the Linux Kernel**. O'Reilly and Associated Inc, 2nd edition, 2003.

[BEBW98] Daniel Frank Moertl Brian Eric Bakke, Frederic Lawrence Huss and Bruce Marshall Walk. Method and apparatus for adaptive localization of frequently accessed, randomly accessed data, June 1998. US Patent No. 5765204, Filed June 5, 1996.

[Bes00] S. Best. Jfs overview: How the journaled file system cuts system restart time to the quick. http://www.ibm.com/developerworks/library/l-jfs.html, January 2000.

[BHK+91] Mary G. Baker, John H. Hartman, Michael D. Kupfer, Ken W. Shirriff, and John K. Ousterhout. Measurements of a distributed file system. In **Proceedings of the Symposium on Operating System Principles**, volume 25, pages 198–212, October 1991.

[Bla92] M. A. Blaze. Nfs tracing by passive network monitoring. In **Proceedings of the Winter USENIX Technical Conference**, pages 333–344, Winter 1992.

[BS02] Eitan Bachmat and Jiri Schindler. Analysis of methods for scheduling low priority disk drive tasks. In **Proceedings of the ACM SIGMOD International Conference on Management of Data**, pages 55–65, 2002.

[DO89] Fred Douglis and John K. Ousterhout. Log-structured file systems. In **COMPCON Proceedings**, pages 124–129, Spring 1989.

[Don88] Shane Mc. Donald. Dynamically restructuring disk space for improved file system performance. Technical report, Department of Computational Science, University of Saskatchewan, Saskatoon, July 1988.

[ELMS03] Daniel Ellard, Jonathan Ledlie, Pia Malkani, and Margo Seltzer. Passive NFS tracing of email and research workloads. In **Proceedings of USENIX Conference on File and Storage Technologies**, March 2003.

[GG97] Jim Gray and Goetx Graefe. The five minute rule ten years later, and other computer storage rules of thumb. In **Proceedings of the ACM SIGMOD International Conference on Management of Data**, 1997.

[GGP99] B. L. Worthington G.R. Ganger and Y.N. Patt. Disksim simulation environment version 3.0. http://www.pdl.cmu.edu/DiskSim, 1999.

[GNA+97] Garth A. Gibson, David F. Nagle, Khalil Amiri, Fay W. Chang, Howard Gobioff, Chen Lee, Berend Ozceri, Erik Riedel, David Rochberg, and Jim Zelenka. File server scaling with network-attached secure disks. In **Proceedings of the ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems**, June 1997.

[GS73] David D. Grossman and Harvey F. Silverman. Placement of records on secondary storage device to minimize access time. volume 20, pages 429–438, July 1973.

[HS03] W. W. Hsu and A. J. Smith. Characteristics of I/O traffic in personal computer and server workloads. **IBM Journal of Research and Development**, 42(2):347–372, 2003.

[Hsu03] Windsor W. Hsu. Dynamic locality improvement techniques for increasing effective storage performance. Technical Report UCB/CSD-03-1223, University of California, Computer Science Divison, 2003.

[JFS] Journaled file system technology for linux. http://oss.software.ibm.com/jfs.

[Knu98] Donald E. Knuth. **The Art of Computer Programming**. Addison Wesley Longman Publishing Co. Inc., Redwood City, CA, 2nd edition, 1998.

[KRK92] P. Biswas K. Ramakrishnan and R. Karelda. Analysis of file I/O traces in commercial computing environments. In **Proceedings of the ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems**, pages 78–90, 1992.

[Mil91] Ethan L. Miller. Input/output behavior of supercomputing applications. Technical Report UCB/CSD 91/616, University of California, Berkeley, January 1991.

[MSC+86] J. H. Morris, M. Satyanarayanan, M. H. Conner, J. H. Howard, D. S. Rosenthal, and F. D. Smith. Andrew: A distributed personal computing environment. **Communications of the ACM**, 29(3), March 1986.

[NC04] Sumit Narayan and John A. Chandy. Trace based analysis of file system effects on disk I/O. In **International Symposium on Performance Evaluation of Computer and Telecommunication Systems**, June 2004.

[OCH+85] John K. Ousterhout, H. Da Costa, D. Harrison, J. A. Kunze, Michael D. Kupfer, and J. G. Thompson. A trace-driven analysis of the UNIX 4.2 BSD file system. In **Proceedings of the Symposium on Operating System Principles**, pages 15–24, December 1985.

[OSD03] Open Source Developement Labs. http://www.osdl.org, 2003.

[RCT94] T. Ts'o R. Card and S. Tweedie. Design and implementation of the Second Extended File System. In **Dutch International Symposium on Linux**, 1994.

[rei] Reiser file system. http://www.namesys.com.

[RLA00] Drew Roselli, Jacob Lorch, and Thomas E. Anderson. A comparison of file system workloads. In **Proceedings of the USENIX Technical Conference**, pages 41–54, June 2000.

[RW91] C. Ruemmler and J. Wilkes. Disk shuffling. Technical Report HPL-91-156, Hewlett-Packard, October 1991.

[RW92] Chris Ruemmler and John Wilkes. UNIX disk access patterns. Technical Report HPL-OSR-92-152, Hewlett-Packard, Palo Alto, CA, 1992.

[RW93] Chris Ruemmler and John Wilkes. A trace-driven analysis of working set sizes. Technical Report HPL-OSR-93-23, Hewlett-Packard, Palo Alto, CA, April 1993.

[Sat81] M. Satyanarayanan. A study of file sizes and functional lifetimes. In **Proceedings of $8^{th}$ ACM Symposium on Operating System Principles**, pages 96–108, 1981.

[Smi85] A. J. Smith. Disk cache-miss ratio analysis and design considerations. **ACM Transactions on Computer Systems**, 3(3):161–203, August 1985.

[SW96] Stefan Savage and John Wilkes. AFRAID – A frequently redundant array of independent disks. In **Proceedings of the USENIX Technical Conference**, pages 27–39, January 1996.

[Tra92] Transaction Processing Performance Council (TPC). **TPC Benchmark C Standard Specification Revision 1.1**. Shanley Public Relations, San Jose, CA, March 1992.

[TT02] T. Ts'o and S. Tweedie. Future directions for ext2/ext3 file systems. In **Proceedings of the USENIX Technical Conference**, June 2002.

[Twe] Stephen Tweedie. http://www.redhat.com.

[UMT] The UMass trace repository. http://traces.cs.umass.edu.

[Ver01] VeriTest. NetBench 7.0.2. "http://www.netbench.com" 2001.

[Vog99] W. Vogels. File system usage in windows nt 4.0. In **ACM Symposium on Operating System Principles**, pages 93–109, December 1999.

[WAP99] Randolph Y. Wang, Thomas E. Anderson, and David A. Patterson. Virtual log based file systems for a programmable disk. In **Proceedings of Symposium on Operating Systems Design and Implementation**, pages 29–43, February 1999.

[Won] C.K. Wong. Minimizing expected head movement in one-dimensional and two-dimensional mass storage systems. In **ACM Computing Surveys**, volume 12, pages 167–178.